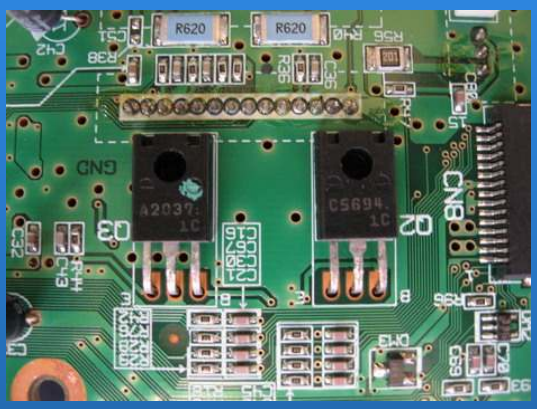
Manual

Electrónica Básica



Electrónica Básica

Electrónica Digital

Microcontroladores

Proyectos

Joan Mengual

Página en blanco

Prólogo

Manual de Electrónica Básica

De Luis:

De principio supongo que tus conocimientos son pocos o nulos en electrónica, y estos serán tus primeros contactos con este mundo.

No será nada difícil...!!!, solo necesitamos unos pocos componentes, aprender a identificarlos, ver la forma adecuada de conectarlos y darles una buena utilidad, pero debo reconocer que lo más importante es disponer de un poco de ingenio, luego se te despertarán las neuronas.

Una cosa más... Tus dudas y consultas quizás sean iguales a las mías, así que veremos como nos la quitamos de encima, finalmente recibe la más calurosa Bienvenida.

De Joan:

No es nada fácil empezar un proyecto sin apenas conocimientos, lo mejor es aprender un poco y después abordarlo, en este libro aprenderás poco a poco a conocer la electrónica y a amarla con tanta pasión que disfrutarás montando proyectos y viendo como lo que construyes con tus manos tienen una utilidad. Si después consigues tener un futuro dedicado a la electrónica y vives de élla podrás comprobar que el tiempo de Luis y el mío no fué en vano

Revisado en Febrero del 2013 y dedicado a Eduardo Vargas Ochoa que espero sea un buen alumno y aprenda esta especialidad que es muy gratificante ya que al tiempo que apredes una profesión de futuro juegas con lo aprendido.

Para todos los lectores:
Ruego me hagan saber los posibles fallos de este libro para corregir, les estaré muy agradecido.

Comenzamos...???

J. Mengual - 2013

Manual de Electrónica Básica

Indice

Electrónica básica ... 8
Símbolos y Componentes ... 8
Protoboard o Placa de Pruebas ... 10
Diodos LED's ... 11
Diodos ... 11
Potenciómetros ... 12
Fotocélula o LDR ... 12
Condensadores o Capacitores Electrolíticos ... 12
Transistores ... 14
El Circuito Integrado NE555 ... 16
Circuitos de conmutación ... 18
Fuente de Tensión Regulada a 5V ... 20
Código de Colores para Resistencias ... 21
Capacitores (Condensadores) Cerámicos ... 22
Combinación de Resistencias ... 23
Motores ... 26
Motores-CC (Motores de Corriente Continua) ... 26
Control de Sentido de Giro para Motores- CC ... 27
Driver para motores ... 28
Aplicación para el control de Motores- CC ... 29
Modulación por Ancho de Pulso (PWM) ... 31
Modulación por Frecuencia de Pulsos (PFM) ... 31
Motores PaP o Motores Paso a Paso ... 32
Descripción general de los motores PaP ... 32
Motores Bipolares ... 33
Motores Unipolares ... 34
Secuencia para Motores-PaP Unipolares ... 36
Software de Prueba para el control de Motores PaP ... 37
Información técnica del L293B ... 39
L293, datos del fabricante ... 39
Elctrónica digital ... 45
Qué es Electrónica Digital? ... 45
Lógica Positiva ... 45
Lógica Negativa ... 45
Compuertas Lógicas ... 46
Compuerta NOT ... 46
Compuerta AND ... 46
Compuerta OR ... 46
Compuerta OR-EX o XOR ... 47
Compuertas Lógicas Combinadas ... 47
Compuerta NAND ... 47
Compuerta NOR ... 47
Compuerta NOR-EX ... 47
Buffer's ... 48
Circuitos Integrados y Circuito de Prueba ... 48

Un poco de Leyes 50
Leyes de De Morgan 50
Más sobre funciones y operadores lógicos 52
Mapas de Karnaugh 53
Circuitos Astables 56
Osciladores, Multivibradores o Circuitos Astables - Parte I 56
Osciladores, Multivibradores o Circuitos Astables - Parte II 58
Osciladores, Multivibradores o Circuitos Astables - Parte III 59
Modulación por ancho de pulso 60
Introducción - ancho de pulso no simétrico 60
Modulación por ancho de pulso Conmutado 60
Demodulación de señales 61
Doblador de frecuencia 61
Circuitos Monoestables 62
Monoestables sencillo 62
Monoestables con dos compuertas NOR 62
Monoestables con dos inversores 63
Cerradura con teclado electrónico 63
Circuitos Biestables 64
Circuitos Biestables - Parte I 64
Circuitos Biestables - Parte II 66
Circuitos Biestables - Parte III 68
Operadores Lógicos en Circuitos Integrados 69
Circuitos Integrados C-MOS y TTL – Datos técnicos 69
Transistores en circuitos de conmutación 73
Introducción 73
Tansistores en circuitos de conmutación 73
Polarización de un transistor NPN como Emisor Común 73
Montajes Darlington 75
Puerto paralelo 77
Introducción 77
Enviando señales por el Registro de Datos 79
Enviando señales por el puerto de control 82
Recibiendo señales por el Puerto Paralelo 83
El puerto Serie 87
El estándar de comunicaciones RS-232 87
Cómo funciona el puerto serie 87
El puerto serie en el PC 88
Sensores Analógicos y Digitales 91
Generalidades 91
Introducción al conversores A/D (Analógico/Digital) 91
Sensores Analógicos 92
Fotocélulas, Fotorresistencias o LDR's 92
Emisión y Recepción de Infrarrojos 95
Sensores Digitales 100
Switchs, Interruptores y Microswitchs, o Bumpers 100
Descripción y uso de algúnos Circuitos Integrados (IC's) 102
Breve Introducción 102
Circuito Integrado NE555 103
Circuito Integrado CD4013 107

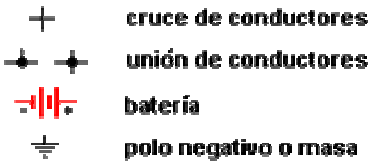
Circuito Integrado CD4017 110
Circuito Integrado CD4029 113
Circuito Integrado CD4066 116
Circuito Integrado CD4511 118
Microcontroladores 120
Conceptos básicos 120
Sistemas microcontroladores 120
Microcontroladores PIC16CXX/FXX de Microchip 121
PIC16C84/F84 122
Capacidad de corriente en los puertos 124
El oscilador externo 124
Reset 125
Arquitectura interna del PIC: 126
Memoria de programa 126
Memoria de datos 127
Programación 128
Configuración de los puertos del PIC 128
El registro STATUS 129
Código para configurar los puertos 129
Primer programa LED1.asm 132
Rutina de retardo 134
Esquema eléctrico para LED1 135
Herramientas de programación 136
Cargando Led1.hex en el PIC 138
Los fusibles de progrmación 140
Los FUSES del PIC con mayor detalle 140
Segundo programa – LED4 142
La rotación 144
Señales de Entrada 145
Programa para verificar el estado de un pulsador 146
Set de Instrucciones 148
Estructura 148
Set de Instrucciones del PIC16F84 149
Instrucciones orientadas a registros: 149
Instrucciones orientadas a bits: 149
Instrucciones orientadas a constantes y de control: 150
Más herramientas - Programadores 151
Interrupciones y temporizadores 152
Interrupciones 152
Fuentes de interrupción en el PCI16F84 152
Rutina de Servicio de Interrupciones (ISR) 154
El Registro OPTION 155
Codificando interrupciones 156
Simulando la interrupción con MPLAB 160
Interrupciones Internas y Temporizaciones 163
Temporizaciones 164
El registro OPTION y el prescaler 165
Cálculo de temporizaciones con el registro TMR0 165
Simulando interrupciones y temporizaciones con TMR0 en MPLAB 169
Más formas de Temporizar 172

Temporizador sin el registro TMR0 173
Un Interesante proyecto 175
Control para la puerta de un garaje 175
Analizando Entradas y Salidas 176
Diagrama de flujo del código principal 177
Diagrama de flujo para el Control de Interrupciones 179
Los esquemas eléctricos 184
Display y tablas 188
Formas de Control 188
Trabajando con un decodificador BCD 189
El Registro PCL 193
Tablas en Asembler: 194
Trabajando Directamente con el Display (Sin Decodificador) 195
Código para el Control del Display sin Decodificador 196
Cuatro Display's 199
Direccionamiento Indirecto 201
Registro 04h (FSR) 201
Registro 00h (INDF) 201
Programa de prueba para 4 display's 203
Código para visualizar 4 display's 206
Proyectos 209
Fuente de Tensión Regulada a 5V 209
Secuenciador de 5 Canales y 2 Efectos 215
Grabador para PIC's 218
Mi Secuenciador Favorito (8 canales) 221
Secuenciador de 32 Canales Controlado por PC 224
Secuenciador de 8 Canales y 4 Efectos 229
Entrenador para el PIC 16F84 232
Emisor y Receptor Infrarrojo 237

Electrónica básica - Primeros contactos

Símbolos y Componentes

Esto para empezar, obviamente no son todos los símbolos y los componentes que existen pero sí los que nos interesan para poder iniciarnos en el tema. Aquí, una breve descripción...



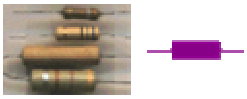
Lo siguiente te servirá de mucho si te acabas de iniciar en este mundillo, a partir de ahora en adelante cada símbolo irá acompañado del aspecto real del componente.

Interruptor (en este caso es un conmutador) No necesita descripción, de todos modos aprende a utilizarlo.

Transformador Otro accesorio. Sólo es un bobinado de cobre, por ahora, nos quedamos con que nos permite disminuir la tensión, en nuestro caso de 220 Volt a 5V, 12V, 24V, etc.

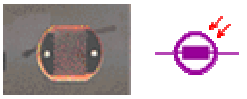
LED (Diodo Emisor de Luz), los hay rojos, verdes, azules, amarillos, también infrarrojos, láser y otros. Sus terminales son ***ánodo*** (terminal largo) y ***cátodo*** (terminal corto).

Diodo Al igual que los LED's sus terminales son ***ánodo*** y ***cátodo*** (este último, identificado con una banda en uno de sus lados), a diferencia de los LED's éstos no emiten luz.



Resistencias o Resistores Presentan una cierta resistencia al paso de la corriente, sus valores están dados en Ohmios, según un **Código de colores** .

Potenciómetros Son resistencias variables, en su interior tienen una pista de carbón y un cursor que la recorre. Según la posición del cursor el valor de la resistencia de este componente cambiará.



Fotocélula También llamada LDR. Una fotocélula es un resistor sensible a la luz que incide en ella. A mayor luz menor resistencia, a menor luz mayor resistencia.

Capacitor de cerámica Estos pueden almacenar pequeñas cargas eléctricas, su valor se expresa en picofaradios o nanofaradios, según un **código establecido**, no distingue sus terminales por lo que no interesa de que lado se conectan.

Condensador ó Capacitor electrólitico Estos almacenan más energía que los anteriores, eso sí, se debe respetar la polaridad de sus terminales. El más corto es el negativo. o bien, podrás identificarlo por el signo en el cuerpo de componente. Si se conectan en polaridad inversa ó una tensión más elevada, tienen la particularidad de explosionar. Mucho cuidado!!!

Transistores Cómo lo digo...! Básicamente un transistor puede controlar una corriente muy grande a partir de una muy pequeña. muy común en los amplificadores de audio. En general son del tipo NPN y PNP, que es eso?, mas adelante se aclararán tus dudas, sus terminales son; **C**olector, **B**ase y **E**misor.

SCR o TIC 106 Son interruptores electrónicos, y se activan mediante un pulso positivo en el terminal G. muy común en sistemas de alarma. Sus terminales son **Á**nodo, **C**átodo y **G**atillo (puerta).

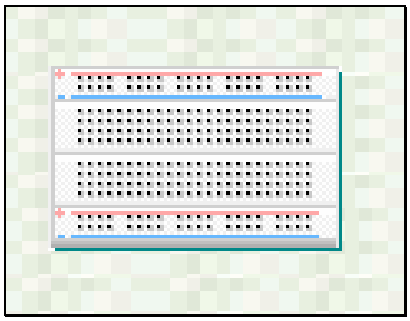
Circuitos Integrados (IC) Un Circuito Integrado (IC) contiene en su interior una gran variedad de componentes en miniatura. Según el IC. de que se trate tendrá distintas funciones o aplicaciones, pueden ser amplificadores, contadores, multiplexores, codificadores, flip-flop, etc. Sus terminales se cuentan en sentido opuesto al giro de las agujas del reloj tomando un punto de referencia.

Relé Básicamente es un dispositivo de potencia, dispone de un electro-imán que actúa como intermediario para activar un interruptor, siendo este último totalmente independiente del electro-imán.

Protoboard o Placa de Pruebas

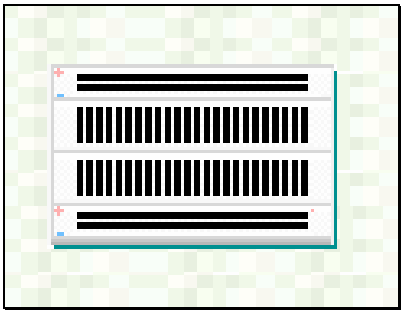
En realidad no necesitas demasiado, de todos modos te mostraré un par de ellas.

Una de las herramienta que utilizaremos de tiempo completo será la placa de pruebas, conocida también como **protoboard**, te permitirá insertar en ella casi todos los componentes siempre y cuando los terminales no dañen los orificios de la misma, de lo contrario no te será de gran ayuda, pero como para todo existe una solución, puedes soldar un cable fino de cobre en los terminales de gran espesor, como en los SCR, los potenciómetros, los interruptores, pulsadores, y otros.



Y aquí está..., en lo posible consigue cables finos y rígidos como los del teléfono para realizar los puentes de unión, son los que más se adaptan a los orificios de la placa, vienen en una gran variedad de colores, los puedes conseguir de 24 hilos de 10, de 8 y bueno... en las casas de electrónica te podrán asesorar.

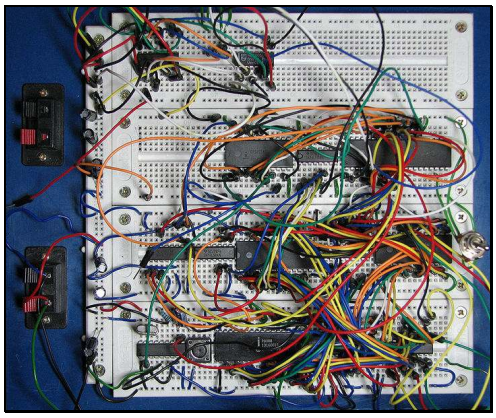
Esto es lo que se encuentra por dentro. las líneas horizontales son las que puedes utilizar para identificar las conexiones a los polos positivo y negativo, fíjate en la imagen anterior que estas líneas están marcadas, con respecto a las verticales, cualquier terminal que conectes en una línea de estas estarán unidos entre sí.



Te habrás dado cuenta que en el medio de las pistas, existe un canal más ancho. Esto se hace para que los chips o integrados puedan calzar adecuadamente en las pistas. Como las dimensiones de los encapsulados están normalizados, cualquier chip que coloques podrás ajustarlo

Los integrados siempre se colocan de esta forma de derecha a izquierda o de izquierda a derecha, como mejor te parezca pero nunca de arriba hacia abajo.

Fíjate en este diseño…. Complicado verdad?



Otra de las herramientas que necesitaras será una batería (esas de 9 volt vienen bien), o con un par de pilas secas bastaría, de todos modos puedes montar tu propia fuente de alimentación

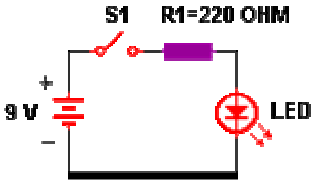
Es importante conseguir un multímetro, multitester o tester, *como lo quieras llamar*, te será de gran utilidad para saber el estado de un componente, si éste se encuentra en condiciones o no, para verificar los fallos en tus circuitos, medir tensiones, resistencias, etc.

Manos a la obra

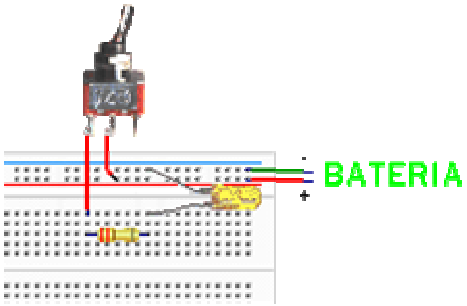
Antes de comenzar quiero aclarar algo... En todas estas prácticas voy a suponer que la corriente eléctrica fluye desde el polo positivo (**+**) hacia el negativo (**-**). Aunque en verdad es a la inversa, de acuerdo?. Esta norma se usa en todo el mundo, no invento nada.

Diodos LED's

El primer circuito, será para ver como encender un LED, recuerda lo de sus terminales, el mas largo (ánodo) apunta al polo (+), el corto (cátodo) al negativo (-), si por alguna razón los terminales son iguales, o lo sacaste de un circuito en desuso, puedes identificar el cátodo por un pequeño corte en la cabeza del componente. **R1** es una resistencia de 220 ohm que hace de protección para el LED, puedes usar otras de mayor valor para ver que ocurre.



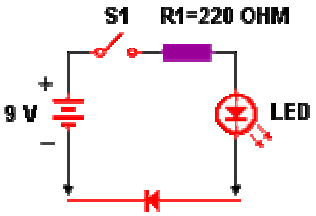
Montado en la placa de prueba, te debería quedar algo así...



Olvidé comentar primero lo de las resistencias, éstas tienen un código de colores que identifica su valor, para verlo ves a la sección de **Código de Colores para Resistencias**, Pág. 21.

Diodos

Los diodos permiten que la corriente circule en un sólo sentido. Un Diodo al igual que un LED necesita estar correctamente polarizado. El cátodo se indica con una banda que rodea el cuerpo del componente.



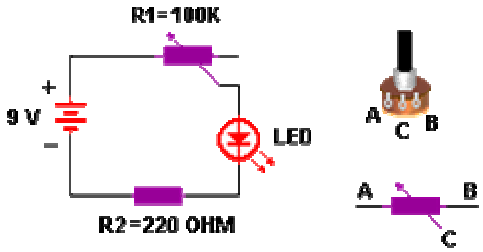
Como no todo está demás podemos utilizar el circuito anterior como un probador de diodos (así de paso vamos armando nuestras propias herramientas).

Según el gráfico el diodo conduce correctamente y el LED se enciende, no así si inviertes el diodo. Su mayor aplicación se encuentra en las **fuentes de alimentación**.

Por cierto el utilizado aquí, es un diodo común del tipo 1N4004, prueba con otros, por ejemplo el 1N4148.

Potenciómetros

Se los encuentra en casi todo aparato electrónico, un ejemplo es el control de volumen de los equipos de audio. En este circuito lo usaremos para controlar el brillo del LED.



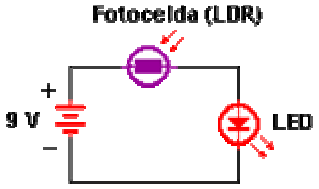
Ahora bien, los extremos A y B del potenciómetro son indistintos ya que la resistencia entre ambos es constante y en nuestro circuito es de 100 k, mientras que la resistencia entre cualquier extremo y el cursor C depende de la posición de este último, pero su máxima resistencia será 100 k.

Si utilizas los contactos A y C, al girar el eje en sentido horario, la resistencia aumentará entre estos dos puntos. Prueba utilizar B y C.

*Te propongo un pequeño desafío...*Intenta armar un circuito con dos LED's de tal modo que al girar el cursor del potenciómetro la intensidad de luz aumente en uno, mientras disminuye en el otro.

Fotocélula o LDR

Muy común en cámaras fotográficas, lo que hacen es mediante el circuito adecuado desactivar el flash cuando hay suficiente luz.



En este ejemplo, *totalmente funcional* si cubres parcial o totalmente la superficie de la fotocelda verás los cambios en el brillo del LED. A más luz incidente, menor será su resistencia, habrá mayor flujo de corriente y mayor será el brillo del LED.

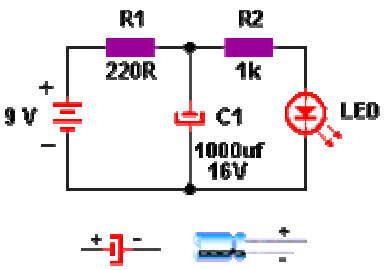
No hay distinción entre sus terminales. Para conseguirla dirígete a cualquier casa de electrónica y pídela como LDR ó fotocélula y elige el tamaño que más te guste.

Condensadores o Capacitores Electrolíticos

Como habrás notado, no haré referencia a los capacitores de cerámica por ahora ya que almacenan muy poca energía de todos modos lo veremos más adelante. Vamos entonces con los Capacitores Electrolíticos.

Estos almacenan más energía que los anteriores, eso sí debes respetar la polaridad de sus terminales. El terminal más corto es el negativo.

Qué pasa si lo saco de un circuito en desuso?. Fácil..., podrás identificarlo por el signo en el cuerpo de componente, *como verás los fabricantes pensaron en todo*.



Montemos el siguiente circuito...

Conectemos la fuente y veamos que ocurre..., *de acuerdo*, no ocurre nada, solo se enciende el LED. Te lo explicaré brevemente.

La corriente que parte de la batería fluye por R1 hacia el nodo, donde se encuentra R2 y el capacitor C1. Aquí comienza a cargarse el Capacitor, una vez cargado, se encenderá el LED, te preguntarás para que me sirve esto?, desconecta la fuente y obtendrás la respuesta.

Si todo va bien, el LED permanecerá encendido por un cierto tiempo gracias a la energía almacenada en el capacitor, a medida que ésta se agote el brillo del LED disminuirá.

Veamos esto un poco más en detalle.

La carga del capacitor depende de su capacidad de almacenamiento, (dado en microfaradios), por otro lado... esa carga se agota a través de R2 o sea que el tiempo de descarga también depende de R2. Así es como llegamos a los conocidos circuitos de tiempo RC (resistencia-capacitor)

Conclusión; la energía almacenada depende del valor de C1, el tiempo en que éste se carga de R1 y el tiempo en que esta energía se agote del producto C.R2. Para interpretarlo mejor, cambia los valores de R1, R2, C1 y luego observa los cambios...

Transistores

Los transistores tienen aplicación en muchísimos circuitos, por lo general son utilizados en procesos de amplificación de señales (las que veremos ahora) y también en circuitos de conmutación a ellos le dedicaremos un lugar especial.

Estos componentes vienen en dos tipos, los NPN y los PNP, no entraré en detalle respecto al nombre ya que podrás notar las diferencias en los circuitos de aplicación, pero sí quiero aclarar algo... Sus terminales...!!! Cada transistor tiene una disposición distinta, según el tipo de que se trate y las ocurrencias de su fabricante, por lo que necesitarás un manual para identificarlos. Uno bastante bueno es el que se encuentra en **www.burosch.de** (*de la mano de su creador...!!!*). Ejecutable en una ventana de DOS, *imperdible...!!!* no requiere instalación, sólo lo descomprimes y ejecutas IC.exe.

Continuemos... veamos ahora estos dos transistores en modo amplificador...

Transistores NPN

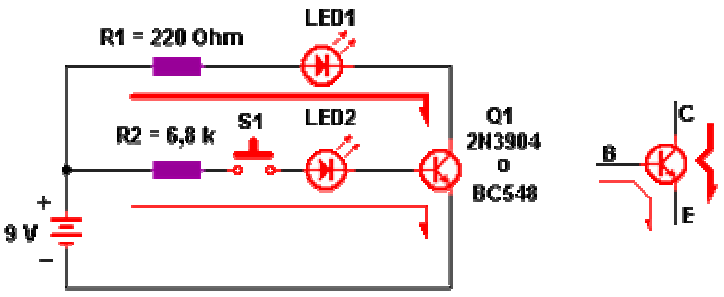
En este ejercicio puedes utilizar uno de los dos transistores que se indican en la siguiente tabla, los dos son del tipo NPN con su respectiva disposición de terminales.

	1	2	3
2N3904	E	B	C
BC548	C	B	E

El circuito que analizaremos será el siguiente...

Cuando acciones **S1** llegará una cierta cantidad de corriente a la base del transistor, esta controlará la cantidad de **corriente** que pasa del Colector al Emisor, lo cual puedes notar en el brillo de los LED's.

Este es el famoso proceso de **AMPLIFICACIÓN.**



Como puedes imaginar, a mayor corriente de base mayor corriente de colector. Prueba cambiar **R2**.

Transistores PNP.

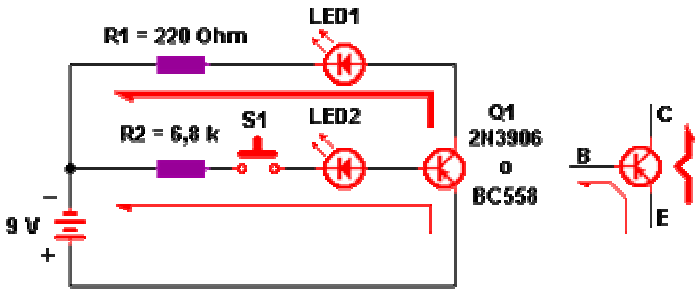
Aquí utilizaremos uno de los dos transistores que se encuentran en el siguiente cuadro.

	1	2	3
2N3906	E	B	C
BC558	C	B	E

En estos transistores, para obtener el mismo efecto que el anterior, su base deberá ser ligeramente negativa.

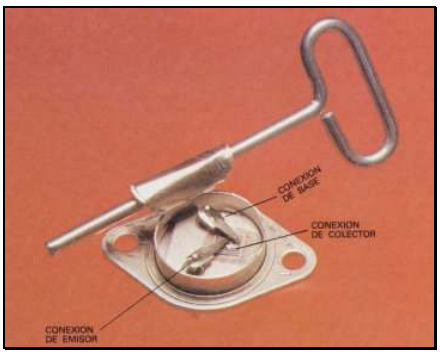
Observa que en este esquema tanto los LED's como la fuente fueron invertidos.

Nuevamente la corriente de base controla la corriente de colector para producir el efecto de AMPLIFICACIÓN.

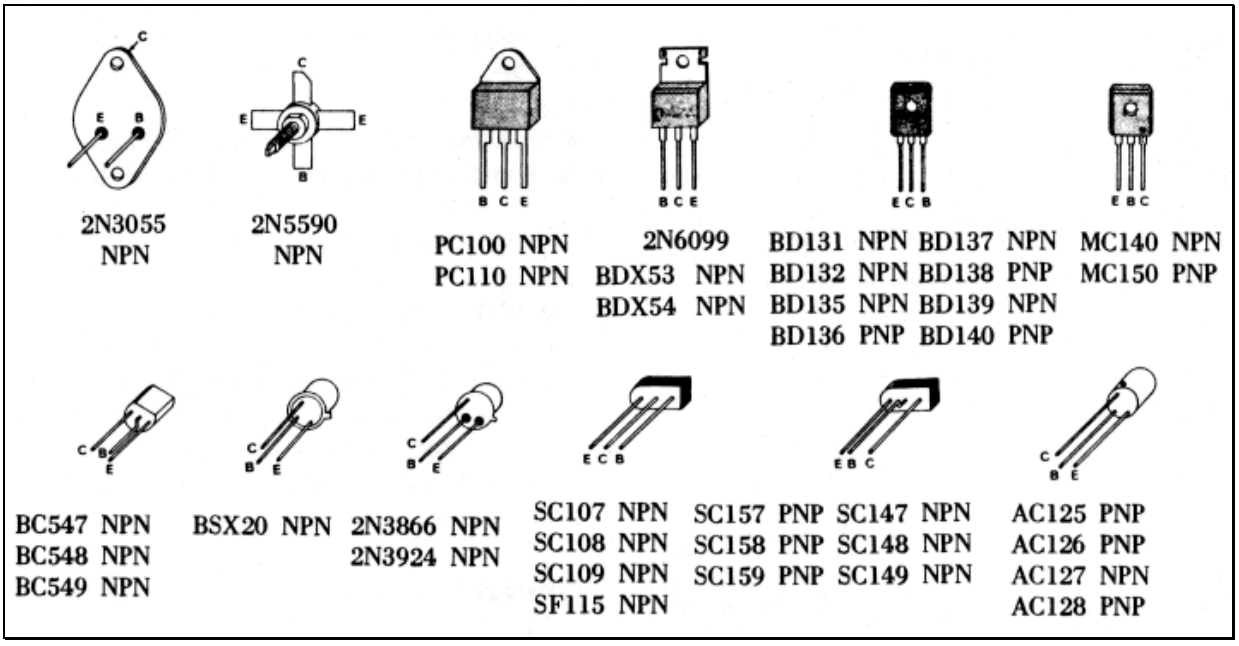


Estarás pensando *¿para qué lo necesito si con el anterior me basta...?*, No es tan así. En muchos casos necesitarás hacer una amplificación y sólo tendrás una pequeña señal ***negativa***. Para entonces, aquí está la solución.

Gráficos de diferentes transistores, entre ellos el más famoso en etapas de potencia, el 2N3055.

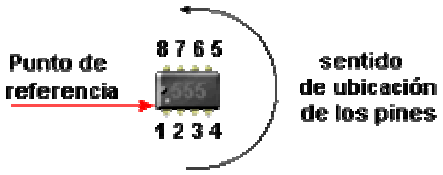


Lo siguiente es una tabla con diversos transistores:



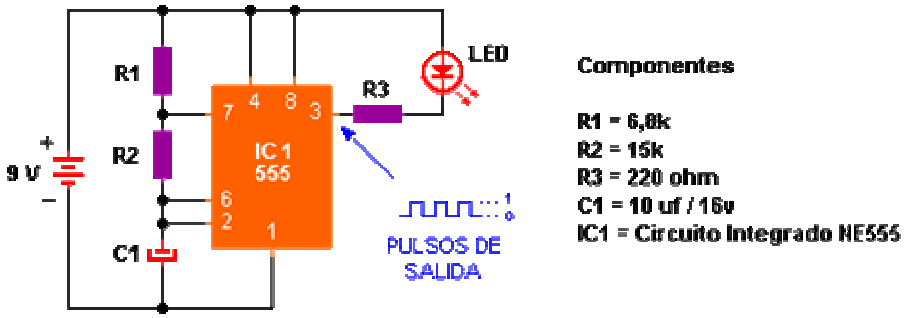
El Circuito Integrado NE555

Esto comienza a ponerse interesantes... Por lo general los esquemas no reflejan la verdadera disposición de sus pines o terminales, así es que para saber cual es el primero y el último observa el siguiente gráfico.



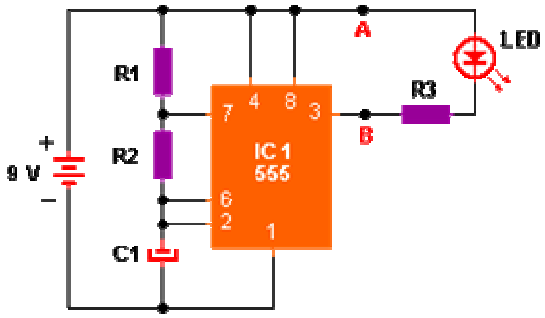
Como verás el integrado en cuestión es un 555, o bien NE555. Se trata de un temporizador (TIMER), utilizado como un generador de pulsos, y la frecuencia de éstos puede variar de 1 pulso por segundo hasta 1 millón de pulsos por segundo, sorprendente verdad?.

Bueno, pero veamos que ocurre aquí; Como necesitamos ver el efecto del circuito le pusimos como siempre un **LED** y una resistencia **R3** conectadas al **pin 3** del 555 (**IC1**), que justamente es el pin de salida.



Observa la polaridad de la fuente respecto al LED..., te habrás dado cuenta que la única forma de encenderlo es que el **pin 3** de **IC1** sea negativo. Y lo será..., observa la onda rectangular de los pulsos de salida..., cuando esté arriba será (+) o 1, y el LED estará apagado. Cuando esté abajo será (-) o 0, entonces el LED se encenderá. Según la señal de salida el LED encenderá de forma alternada.

Veamos los otros componentes; R1, R2 Y C1 forman una red de tiempo. El capacitor **C1** se cargará a través de **R1** y **R2**, del otro lado el 555 espera impaciente que termine de hacerlo, y cuando lo logre lo reflejará en su terminal de salida **(pin 3)**, y he aquí el pulso que produce la descarga del capacitor. Ahora sí..., ya estamos listos para la siguiente carga que generará el segundo pulso. Veamos que modificaciones podemos hacerle al circuito.

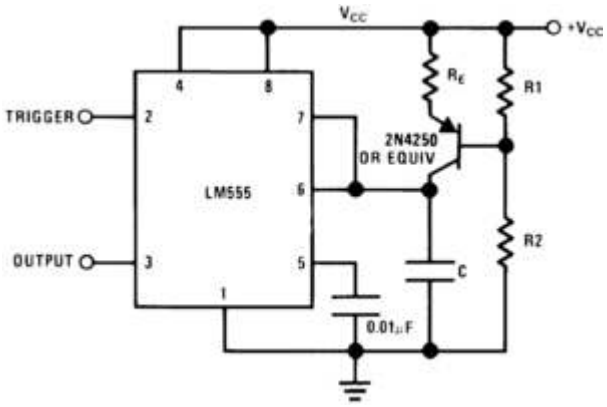


En este esquema marqué los puntos A y B, allí puedes conectar un pequeño parlante (como los de PC), ahora cambia C1 por un capacitor de cerámica (el que tengas a mano, cualquiera va bien), intercala un potenciómetro de 100k entre R2 y el pin 6. Si haces esto obtendrás un generador de sonido.

Otra cosa que puedes hacer es agregarle otra resistencia igual a R3 y un LED más entre los puntos B y el polo negativo de la fuente, pero invertido, y obtendrás algo así como un semáforo, *claro...* si un LED es rojo y el otro verde.

En fin, son muchos los cambios que le puedes hacer y los resultados obtenidos son muy llamativos.

A continuación tenemos un esquema teórico y del patillaje del 555:



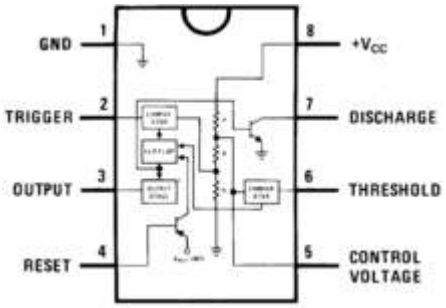
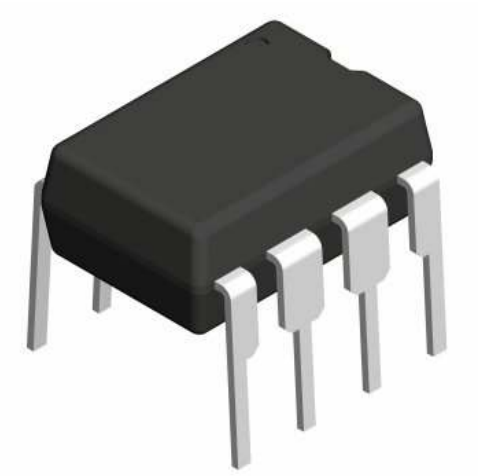


Diagrama físico del 555



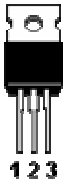
Circuitos de conmutación

SCR ó Tiristor

En nuestro caso es un TIC 106.

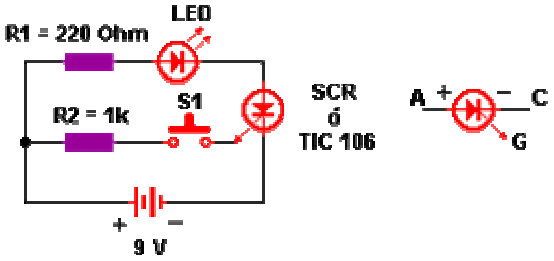
Es un dispositivo electrónico que tiene dos estados de funcionamiento: conducción y bloqueo. Posee tres terminales: Anodo (A), Cátodo (K) y Gate (G, también llamado puerta en español) .

Ya sé que me repito, pero es importante que lo conozcan, son dispositivos sólidos de conmutación (es decir, no son mecánicos) y sus terminales C, Á y G, estan distribuidos según el siguiente cuadro.



	1	2	3
TIC 106D	C	A	G

El SCR es una llave electrónica, que se activa cuando se aplica un pequeño voltaje positivo a su compuerta **G** (gatillo). No creas tan fielmente en todo lo que yo digo, monta el circuito y pruébalo.

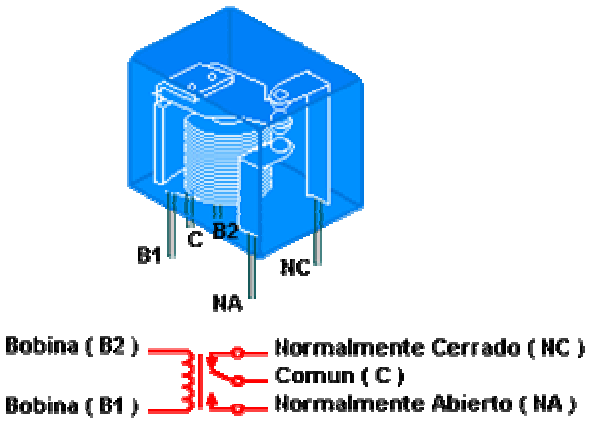


Lo interesante aquí es que una vez disparado el SCR, éste conducirá de forma permanente (si la corriente que ingresa por el ánodo es continua), para desactivarlo sólo quita la fuente de alimentación, conéctalo de nuevo y estará listo para un nuevo disparo.

Cambia el valor de R2 para conocer los límites de sensibilidad del SCR..

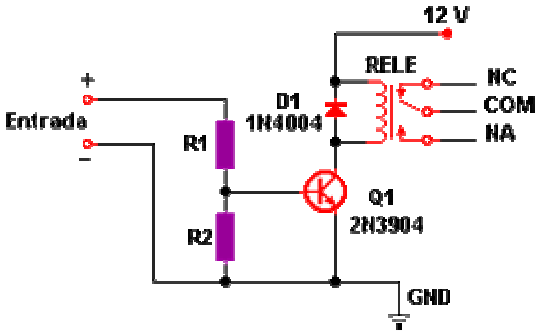
El Relé.

Te diré algo... Todo circuito que construyas y te permita encender un LED también te permitirá encender cualquier aparato eléctrico de casa, como una lámpara por ejemplo, y eso es justamente lo que haremos ahora, en el siguiente gráfico tienes un relé de 5 terminales...



B1 y **B2** son los terminales de alimentación de la bobina, cuando circule corriente por ellos el relé se activará cambiando de posición su interruptor interno y el terminal **C** se conectará con el terminal **NA**.

Veamos ahora un circuito de aplicación...



La señal que le des en la entrada por el extremo **(+)** pasara por **R1** a la base de **Q1** que es un transistor NPN y este pasará a conducir accionando el relé, **D1** esta para compensar la inducción de la bobina, **R2** mantiene el transistor en corte cuando no existe señal alguna por la entrada, su valor es igual al de R1 de 2,7k o puede ser de 2k2 si Q1 es del tipo BC548 o BC337, el relé utilizado debe ser acorde a la alimentación del circuito, en este caso de 12V, puedes utilizar uno de 6V y entonces alimentar al circuito con 6V.

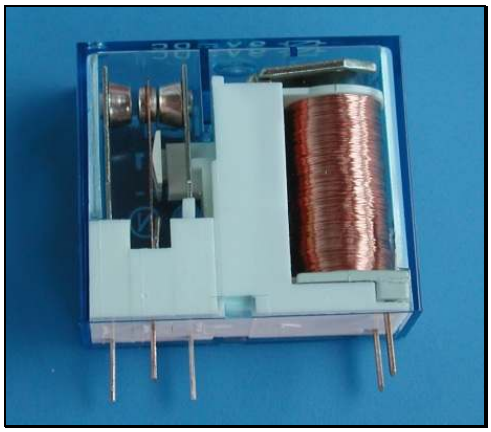
Para conectar la lámpara al circuito hazlo del siguiente modo...

Ahora vamos a combinar los circuitos vistos hasta el momento...

Recuerdas el esquema del 555...?, los puntos A y B...?, bien, conecta la entrada del esquema de relé en esos puntos, (A al (+), y B al (-)), luego conecta el esquema de la lámpara al relé, verifica que todo esté en orden y alimenta el circuito.

sorpresaaaaa...!!! hemos construido una lámpara psicodélica, aunque para conmutaciones rápidas y de potencia usaremos tiristores y en caso de controlar la corriente alterna serán los Triacs.

Foto de un relé con contactos conmutado:



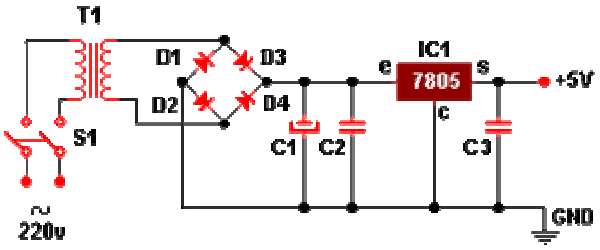
Fuente de Tensión Regulada a 5V

Como las comúnmente llamadas pilas o baterías suelen agotarse en muy corto tiempo es bueno armarse con una de ellas.

¿Qué necesitas para comenzar?...En primer lugar un transformador, que lo puedes extraer de cualquier aparato en desuso, 4 diodos en buenas condiciones, unos cuantos capacitores, y lo que seguramente no encontrarás por allí es un regulador de tensión, estos últimos mantienen un nivel de tensión estable a 5V, 9V, 12V, etc.

Existen los reguladores de tensión positivos y los negativos, se distinguen fácilmente por su nombre. Los primeros corresponden a la serie 78XX y los negativos a la serie 79XX, donde XX es el voltaje a regular. Veamos un ejemplo; si deseas regular la tensión a +5V utilizarás un 78**05**, si deseas hacerlo a +9V acudirás a un 78**09**, y si deseas +12V, bueno... un 78**12**, fácil verdad?...

Aquí está el esquema eléctrico de una fuente regulada a +5V...



Para comprender mejor el proceso de rectificación de la fuente, lo vamos a dividir en varias etapas...

La fuente y su funcionamiento queda explicada ampliamente en el apartado de proyectos, aquí expongo el esquema para que el lector empiece a acostumbrarse a un circuito que será muy habitual en cualquier aplicación electrónica, la fuente puede ser sencilla como el esquema que acabp de presentar o muy compleja, analógica o conmutada, etc.

Código de Colores para Resistencias

El valor de los resistores se puede identificar por los colores de las 4 bandas que rodean al componente, una de ellas es llamada tolerancia, es algo así comooooo... el error de fabricación, esta banda puede ser dorada o plateada, *yo utilizaré la dorada*. La pregunta es . ¿Cómo se leen las otras tres...?

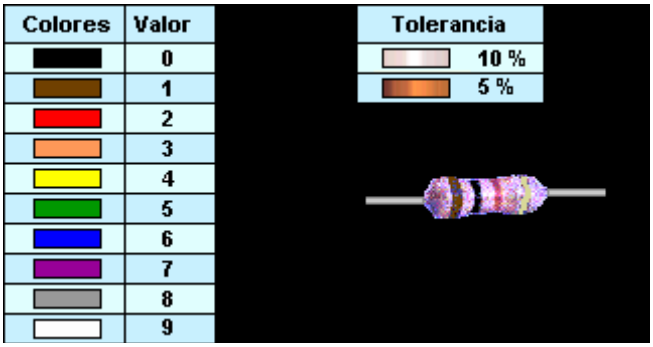


Tabla básica del código de colores

Lo describiré con un ejemplo:

Veamos el valor de este resistor;

La primer banda es el primer dígito y es **marrón=1**,
la segunda es el segundo dígito **negra=0**
y la tercera es la cantidad de ceros **roja=dos ceros**.

Entonces su valor será: **1000** ohm o sea 1 kilo o 1k, si tendría 1000000, seria 1 Mega o 1M.

Es decir que para una resistencia de 70 ohm sus colores deberían ser violeta, negro y negro.

Existen casos en los cuales necesitamos un resistor de un valor determinado y no disponemos de él, la solución es combinar o unir resistores de otros valores de tal modo de obtener el que estamos buscando, para mayor detalle consulta el apartado "**Combinación de resistencias**".

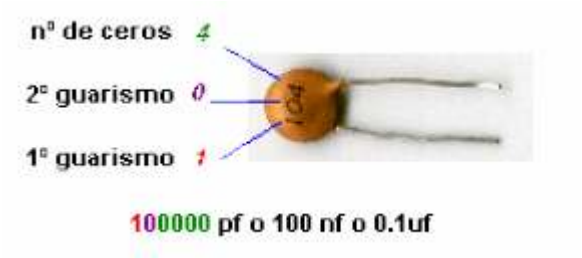
Código de colores en las resistencias

COLORES	Banda 1	Banda 2	Banda 3	Multiplicador	Tolerancia
Plata				x 0.01	10%
Oro				x 0.1	5%
Negro	0	0	0	x 1	
Marrón	1	1	1	x 10	1%
Rojo	2	2	2	x 100	2%
Naranja	3	3	3	x 1000	
Amarillo	4	4	4	x 10000	
Verde	5	5	5	x 100000	0.5%
Azul	6	6	6	x 1000000	
Violeta	7	7	7		
Gris	8	8	8		
Blanco	9	9	9		
--Ninguno--	-	-	-		20%

Capacitores (Condensadores) Cerámicos

Código de valores para Capacitores cerámicos.

a) En algunos casos el valor esta dado por tres números...

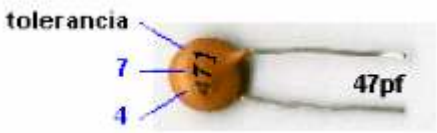


1º número = 1º guarismo de la capacidad.
2º número = 2º guarismo de la capacidad.
3º número = multiplicador (número de ceros)

La especificación se realiza en picofarads.

Ejemplo:
104 = 100.000 = 100.000 picofarad ó = 100 nf

b) En otros casos esta dado por dos números y una letra mayúscula.



Igual que antes, el valor se da en picofarads

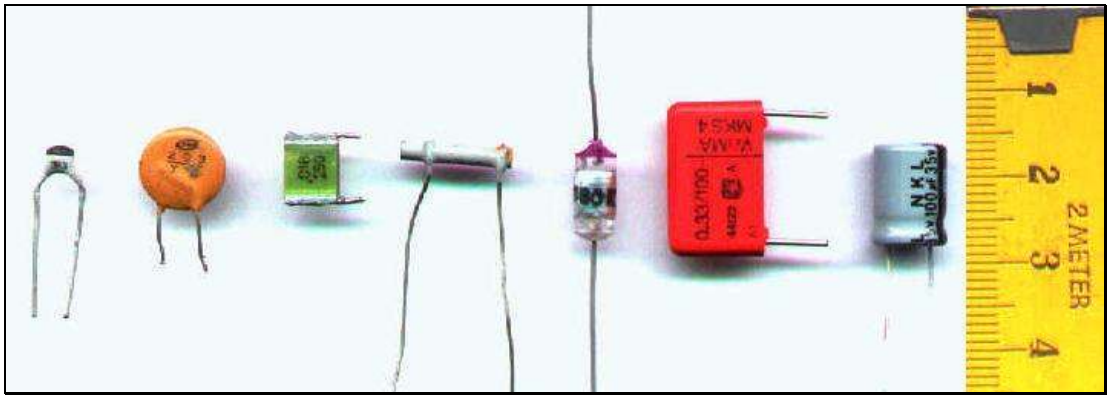
Ejemplo:
47J = 47pF, 220M = 220pF

Para realizar la conversión de un valor a otro, te puedes guiar por la siguiente tabla...

CONVERSION DE UNIDADES

Para convertir	en	Multiplique por
picofarad	nanofarad	0.001
picofarad	microfarad	0.000.001
nanofarad	microfarad	0.001
microfarad	nanofarad	1.000
nanofarad	picofarad	1.000
microfarad	picofarad	1.000.000

Imágenes de diferentes condensadores:



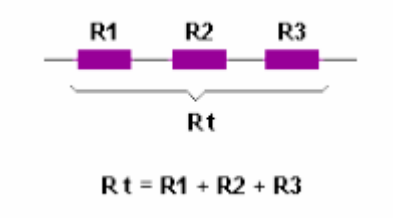
Combinación de Resistencias

La unión de resistencias la podemos hacer de dos maneras, ya sea en un circuito en serie o en paralelo y a la vez una combinación de ambos.

Veamos algunos ejemplos...

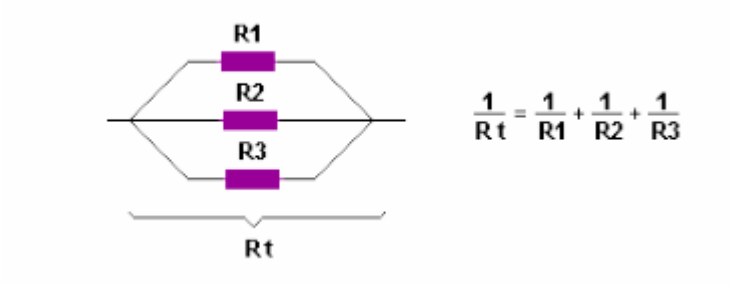
Resistencias en Serie...

En un circuito en serie las resistencias se colocan una seguida de la otra de tal modo que la corriente deberá fluir primero por una de ellas para llegar a la siguiente, esto implica que el valor de la resistencia total del circuito sea la suma de todas ellas.



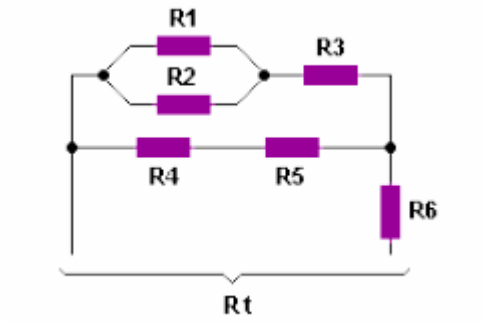
Resistencias en Paralelo...

En un circuito en paralelo las resistencias se colocan según se indica en el siguiente grafico, de esta manera la corriente eléctrica llega a todas las resistencias a la vez, aunque la intensidad de la corriente es mayor por el resistor de menor valor. En este caso la resistencia total del circuito la puedes obtener utilizando la ecuación que se muestra en el grafico...



Circuitos Combinados...

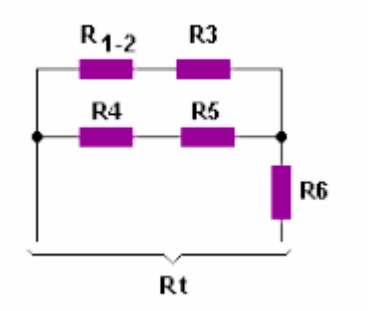
Hay casos en que se combinan resistencias en serie y en paralelo a la vez, estos son llamados circuitos combinados, y para obtener el valor total de la resistencia se resuelve separándolos en mallas. Observa el siguiente circuito...



Podemos comenzar por los circuitos mas sencillos como resolver **R 1-2**, que representa la resistencia total entre R1 y R2, como están en paralelo...

1/R 1-2 = 1/R1 + 1/R2

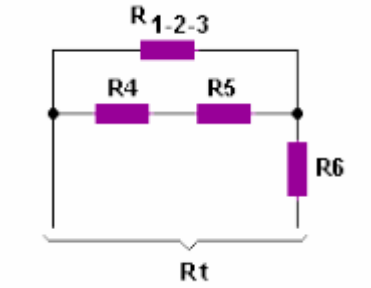
En estos momentos tenemos resueltos R1 y R2 y el circuito nos queda como se ve a continuación...



Combinando el resultado anterior con R3 y teniendo en cuenta que se trata de un circuito en serie...

R 1-2-3 = R 1-2 + R3

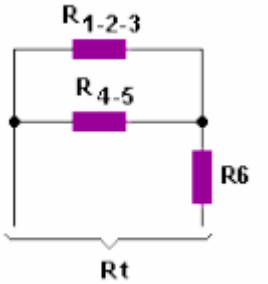
y el circuito nos va quedando mas pequeño, algo así...



Nuevamente tenemos un circuito en serie entre R4 y R5, entonces...

R 4-5 = R4 + R5

De tal modo que la suprimimos y la reemplazamos por **R 4-5**.

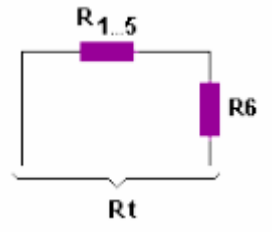


Te habrás dado cuenta que cada vez la malla de nuestro circuito se va reduciendo, sucede que es una forma sencilla resolverlo por pasos, con la practica no necesitaras hacerlo ya que puedes resolverlo mentalmente.

Pero continuemos..., Ahora resolvemos el circuito en paralelo para obtener **R 1...5**

1/R 1...5 = 1/R 1-2-3 + 1/R 4-5

Finalmente obtuvimos el circuito más sencillo de todos y es un circuito en serie el cual nos da la resistencia total...



y el cálculo final seria como sigue...

Rt = R 1...5 + R6

Estos ejemplos para cualquier situación que se te pudiera presentar, no viene nada mal verdad?.

Motores CC y Motores PaP.

Bueno, primero lo primero...

En esta sección trataré de describir algunos tipos de motores, en especial aquellos que son de mayor uso, como los Motores de Corriente Continua y los Motores PAP (paso a paso).

Claro que este tema ya está muy difundido en la red (www), y es posible encontrar tutoriales muy, pero muy buenos, de los cuales yo aprendí, así que imagínate si los hay...

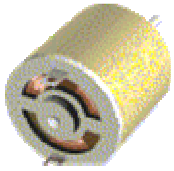
A continuación describo los dos tipos de motores

* Motores-CC (Motores de Corriente Continua) a continuación….
* Motores PaP (Motores Paso a Paso) un poco más adelante...

Motores

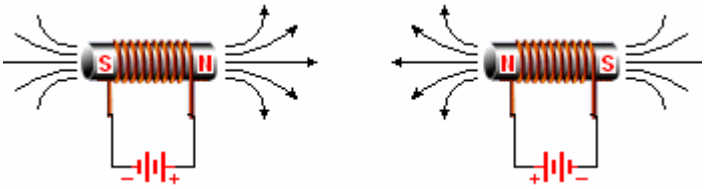
Motores-CC (Motores de Corriente Continua)

Son de los más comunes y económicos, y puedes encontrarlo en la mayoría de los juguetes a pilas, constituidos, por lo general, por dos imanes permanentes fijados en la carcaza y una serie de bobinados de cobre ubicados en el eje del motor, que habitualmente suelen ser tres.

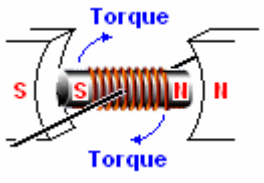


El funcionamiento se basa en la interacción entre el campo magnético del imán permanente y el generado por las bobinas, ya sea una atracción o una repulsión hacen que el eje del motor comience su movimiento, bueno, eso es a grandes rasgos...

Ahora nos metemos un poco más adentro... Cuando una bobina es recorrida por la corriente eléctrica, esta genera un campo magnético y como es obvio este campo magnético tiene una orientación, es decir dos polos un polo NORTE y un polo SUR, la pregunta es, cuál es cuál...?, y la respuesta es muy sencilla, si el núcleo de la bobina es de un material ferromagnético los polos en este material se verían así...



como puedes ver, estos polos pueden ser invertidos fácilmente con sólo cambiar la polaridad de la bobina, por otro lado, al núcleo de las bobinas las convierte en un electroimán, ahora bien, si tienes nociones de el efecto producido por la interacción entre cargas, recordarás que cargas opuestas o polos opuestos se atraen y cargas del mismo signo o polos del mismo signo se repelen, esto hace que el eje del motor gire produciendo un determinado torque



Te preguntarás que es el torque...?, pues es simplemente la fuerza de giro, si quieres podríamos llamarle la potencia que este motor tiene, la cual depende de varios factores, como ser; la cantidad de corriente, el espesor del alambre de cobre, la cantidad de vueltas del bobinado, la tensión etc. esto es algo que ya viene determinado por el fabricante, y que nosotros poco podemos hacer, más que jugar con uno que otro parámetro que luego describiré.

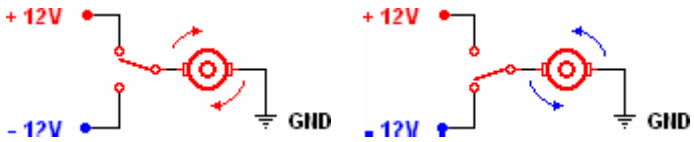
La imagen anterior fue solo a modo descriptivo, ya que por lo general suelen actuar las dos fuerzas, tanto atracción como repulsión, y más si se trata de un motor con bobinas impares.

Estos motores disponen de dos bornes que se conectan a la fuente de alimentación y según la forma de conexión el motor girará en un sentido u otro, veamos eso justamente.

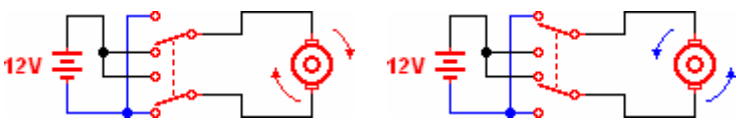
Control de Sentido de Giro para Motores- CC

Existen varias formas de lograr que estos motores inviertan su sentido de giro, una es utilizando una fuente simétrica o dos fuentes de alimentación con un interruptor simple de dos contactos y otra es utilizar una fuente común con un interruptor doble es decir uno de 4 contactos, en todos los casos es bueno conectar también un capacitor en paralelo entre los bornes del motor, éste para amortiguar la inducción que generan las bobinas internas del motor (aunque no lo representaré para facilitar un poco la comprensión del circuito, está...?), las conexiones serían así...

Con Fuente Simétrica o Doble Fuente



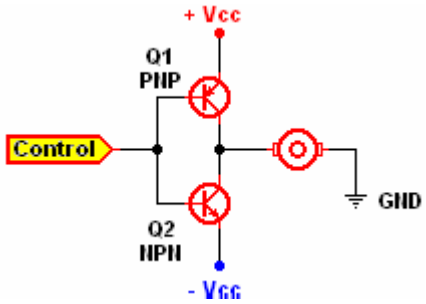
Con una Fuente Simple



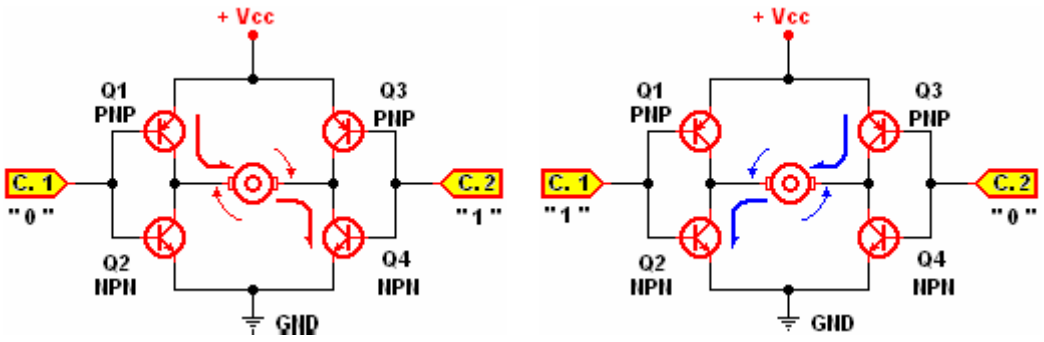
Otra solución cuando intentas que uno de tus modelos realice esta tarea por su propia cuenta, es sustituir los interruptores por los relés correspondientes e idearte un par de circuitos para lograr el mismo efecto.

Aunque esta última opción es una de las más prácticas, tiene sus inconvenientes ya que los relés suelen presentar problemas mecánicos y de desgaste, lo ideal sería disponer de un circuito un poco más sólido, quitando los relés y haciendo uso de transistores, estos últimos conectados en modo corte y saturación, así actúan como interruptores, un análisis más completo de esta forma de conexión la puedes ver en la sección de transistores de conmutación.

Veamos como hacerlo con una fuente simétrica... En este caso será necesario el uso de dos transistores complementarios es decir uno PNP y otro NPN, de este modo sólo es necesario un terminal de control, el cual puede tomar valores lógicos "0" y "1", el esquema de conexiones es el que sigue...



Cuando intentas utilizar una fuente de alimentación simple la cosa se complica un poco más, pero como todo tiene solución lo puedes implementar del siguiente modo.



Estos circuitos son conocidos como puente en H o H-Bridge, en realidad son más complejos de lo que describí aquí, pero esta es la base del funcionamiento de los **Drivers** para motores.

Ahora bien, estos Driver's que acabo de mencionar son circuitos integrados que ya traen todo este despiole metido adentro, lo cual facilita el diseño de nuestros circuitos, tales como el UCN5804, el BA6286, el L293B, L297, L298 o también puedes ingeniártelas con el ULN2803 o el ULN2003, estos dos últimos son arrays de transistores, pero apuesto que te las puedes arreglar.

Veamos como trabajar con el integrado L293B...

Driver para motores

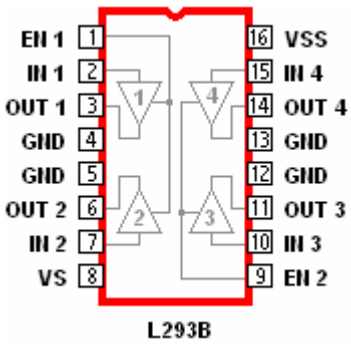
Entre los Drivers mencionados anteriormente, el más conocido es el L293B. Toda la información aquí disponible fue extraída de la hoja de datos de este integrado, que puedes bajarlo desde la web del fabricante, aunque claro..., está en inglés.

Bueno, se trata de un Driver para motores de 4 canales, y observa las ventajas que tiene...

- Cada canal es capaz de entregar hasta 1A de corriente.

- Posee una entrada de alimentación independiente que alimenta los 4 Drivers, es decir la que requieren los motores...
- El control de los Drivers es compatible con señales TTL es decir con 5 voltios (estamos hablando de señales lógicas).
- Cada uno de los 4 Drivers puede ser activado de forma independiente (por su terminal de entrada), o habilitado de dos en dos con un sólo terminal (Enable).

Aquí una imagen del integrado y su tabla de verdad para que la analices...



IN_n	OUT_n	EN_{1-2}
H	H	H
L	L	H
H	X	L
L	X	L

En esta tabla de Verdad la entrada **EN_{1-2}** habilita dos de los canales de este integrado cuando se encuentra a nivel **H** (alto), de tal modo que la salida **OUT_n** tomará el valor de la entrada **IN_n**.

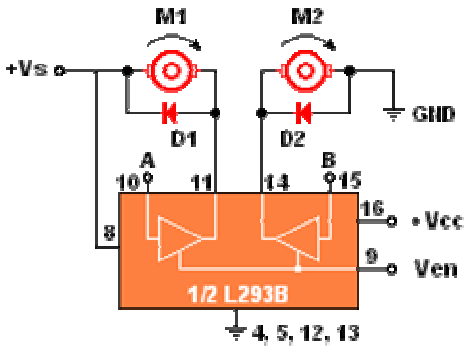
Por otro lado **OUT_n** quedará en alta impedancia (**X**) si el terminal **EN_{1-2}** se encuentra a nivel bajo (**L**), es decir que en este caso ya no tiene importancia el valor de **IN_n**, y por lo tanto **OUT_n** quedará totalmente libre.

Por último, aclarar que VS y VSS son los pines de alimentación, **VS** para los 4 Drivers que según el Datasheet puede tomar valores desde VSS hasta 36V, y **VSS** es el terminal de alimentación TTL, para nosotros sería como **+VCC**, se entiende...???

Este integrado no sólo tiene aplicación en Motores-CC sino también en Motores-PaP pero ese tema lo tocaremos a su debido momento...

Aplicación para el control de Motores- CC

Tenemos dos posibilidades de control, una es controlar los motores en un sólo sentido de giro, es decir, hacer que gire o detenerlo, en este caso tienes posibilidad de controlar hasta 4 motores, veamos el esquema del circuito en cuestión...



Como verás aquí sólo se representa la mitad del integrado, la otra mitad es lo mismo, sólo cambia el número de los pines.

Con los terminales **A** y **B** controlas el encendido del motor correspondiente, Con **Ven** habilitas o no los terminales de control A y B, recuerda que **Ven** debe estar en nivel alto si quieres utilizar los terminales de control A y B. Finalmente la forma de control sería como se ve en la siguiente tabla.

Ven	A	M1	B	M2
H	H	Detiene	H	Gira
H	L	Gira	L	Detiene
L	X	Parada Libre	X	Parada Libre

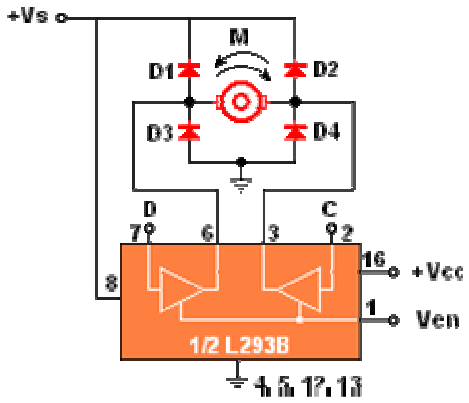
+Vcc es el terminal de alimentación compatible con la señal de control A y B, o sea +5V, **Vs** tiene los niveles de tensión requeridos por el motor (12, 15, 20, hasta 36v).

D1 y D2, bueno... como los capacitores... es para proteger al integrado de las tensiones generadas por la inducción de las bobinas del motor.

Una cosa más que quiero hacer notar... Observa que un motor (M1) esta unido a +Vs, mientras que el otro (M2) esta a GND, puedes utilizar cualquiera de las dos configuraciones, aquí las grafiqué tal como están en la hoja de datos, la cual de por sí, es demasiado clara, e intenta mostrar todas las posibilidades de conexión...

Bien, pasemos al segundo método de control...

Este es el que más me gusta, pero requiere dos de los 4 driver del integrado, la forma de conexión seria como sigue...



Ejemmmmm, creo que quedará más claro si analizamos la tabla de verdad de este circuito, así evito perderme...

Ven	C	D	Función del motor
H	H	L	Giro a Derecha
H	L	H	Giro a Izquierda
H	C=D		Detención
L	X	X	Motor Libre

Esta tabla indica claramente como puedes controlar el motor, en un sentido o en otro, detenerlo o dejarlo libre, esto último de dejarlo libre se refiere a que cualquier señal en los terminales de control A, B, C y D no tendrán efecto alguno sobre el motor, es decir que ha quedado liberado totalmente.

Como ves tienes muchas posibilidades para comandar estos motores usando el integrado L293B.

Bueno, no es una locura de información pero creo que de algo les puede servir.

Otro de los aspectos que llaman la atención en los Motores-CC es el control de velocidad. Los circuitos anteriores están de maravilla, pero nada dicen de este tema, aunque queda claro que esto se debe llevar a cabo por los terminales de control A, B, C y D, por lo tanto necesitamos un circuito aparte, veamos de que se trata esto...

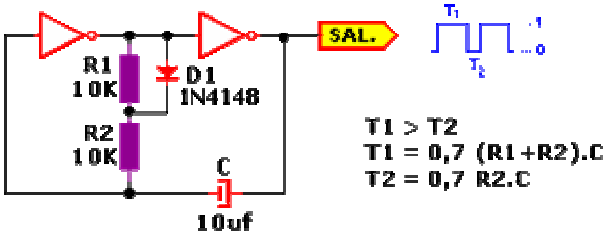
CONTROL DE VELOCIDAD DE MOTORES

Modulación por Ancho de Pulso (PWM)

En la mayoría de los tutoriales que encontré en la red se recomienda el control por Modulación de Ancho de Pulso conocido como PWM, un tema que ya vimos en el tutorial de electrónica digital y que básicamente consiste en enviar a los terminales de control un tren de pulsos los cuales varían en mayor o menor medida el tiempo en que se mantiene el nivel alto de esos pulsos, manteniendo la frecuencia constante, así...

Con esta forma de control la cantidad de corriente sigue siendo la misma, la tensión no varía y en consecuencia el torque del motor se mantiene, que es justamente lo que estábamos buscando.

Un circuito de ejemplo puede ser el que ya mencionamos en el tutorial de electrónica digital que es algo así...



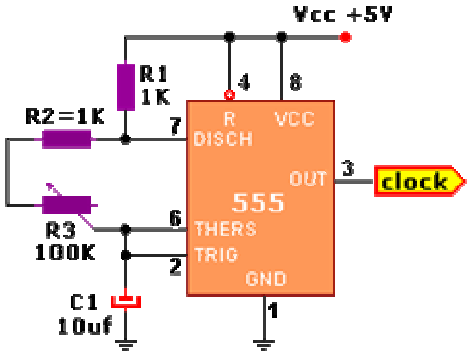
En el cual puedes reemplazar R1 por un potenciómetro y así controlar los tiempos de los niveles de salida...

Respecto a esta clase de circuitos se puede encontrar muchos por la red, solo busca "modulación por ancho de pulso" o "PWM", y tendrás para divertirte, y si desean compartir sus circuitos, pues bienvenido sean, se los cargaré sin problemas, en fin...

Modulación por Frecuencia de Pulsos (PFM)

Creo que el título lo dice todo, se trata de eso mismo, variar la frecuencia de pulso en los terminales de control, lo cual puedes lograr fácilmente con un circuito astable que bien podría ser un 555, y utilizar un potenciómetro para variar la frecuencia de los pulsos, el efecto que lograrías en comparación al anterior sería algo así...

Claro que para mayor velocidad la frecuencia de los pulsos iría mucho más rápida de lo que se muestra en esta imagen. El esquema para el 555 podría ser el que sigue...



Si estos motores fueran controlados digamos por un microcontrolador, la cosa sería mucho más sencilla ya que podrías tener mayor control sobre el circuito, respecto a eso lo dejo librado a tu imaginación, supongo que con esto tienes suficiente.

Motores PaP o Motores Paso a Paso

Descripción general de los motores PaP

A diferencia de los Motores-CC, que giran a lo loco cuando son conectados a la fuente de alimentación, los Motores-PaP solamente giran un ángulo determinado, los primeros sólo disponen de dos terminales de conexión, mientras los otros pueden tener 4, 5 o 6, según el tipo de motor que se trate, por otro lado los motores de corriente continua no pueden quedar enclavados en una sola posición, mientras los motores paso a paso sí.

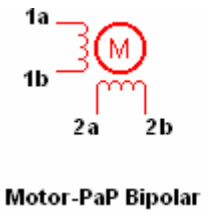
Esas son sólo algunas de las diferencias entre ambos tipos de motores, el primer contacto que tuve con uno de estos fue cuando desarmé una disquetera de esas antiguas de 5 1/4, y la pregunta era "como ponerlo en funcionamiento...???" hasta que encontré muy buena información al respecto, y aquí vamos...

Los motores paso a paso son comúnmente utilizados en situaciones en que se requiere un cierto grado de precisión, por ejemplo en las disqueteras anteriormente mencionada puedes encontrarlo unido al cabezal haciéndolo avanzar, retroceder o posicionarse en una determinada región de datos alojadas en el disquete.

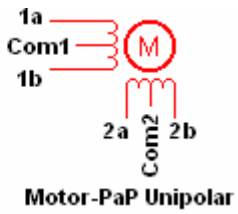
El ángulo de giro de estos motores es muy variado pasando desde los 90º hasta los 1.8º e incluso 0.72º, cada ángulo de giro, (también llamado paso) se efectúa enviando un pulso en uno de sus terminales, es decir que por ejemplo en motores que tienen 90º de giro por paso, se requiere 4 pulsos para dar una vuelta completa, mientras que en los de 1,8º necesitas 200 pulsos, y en los otros necesitas 500.

El que tengo aquí a mano tiene un ángulo de giro de 3,6º es decir que necesita 100 pulsos para dar una vuelta completa, esto puede variar según la forma de control que quieras utilizar.

Los Motores-PaP suelen ser clasificado en dos tipos, según su diseño y fabricación pueden ser Bipolares o Unipolares, aquí tienes un par de imágenes que lo diferencian el uno del otro...



Motor-PaP Bipolar



Motor-PaP Unipolar

Espero que hayas notado la diferencia, bueno, es que mientras los Unipolares disponen de dos bobinas independientes los Bipolares parecieran tener 4 debido al terminal central que es el común de cada par de bobinas, pues a eso se debe aquello de los 6 cables y que si unes los terminales Com1 y Com2 tienes un terminal común y 4 terminales de control (es decir 5 cables). Bien, ahora veamos como controlar estos motores...

Motores Bipolares

Si tienes la suerte de toparte con uno de estos,(el que tengo yo lo saqué de un disco duro, :op) deberás identificar los cables 1a, 1b, 2a y 2b, lo cual es muy sencillo, ya que si utilizas un tester puedes medir la resistencia entre cada par de terminales, ya que los extremos 1a y 1b deben tener la misma resistencia que los extremos 2a y 2b, ahora si mides la resistencia en forma cruzada no te marcará nada ya que corresponden a bobinas distintas.

Bien, pasemos ahora a lo más interesante que es controlar estos motores bipolares.

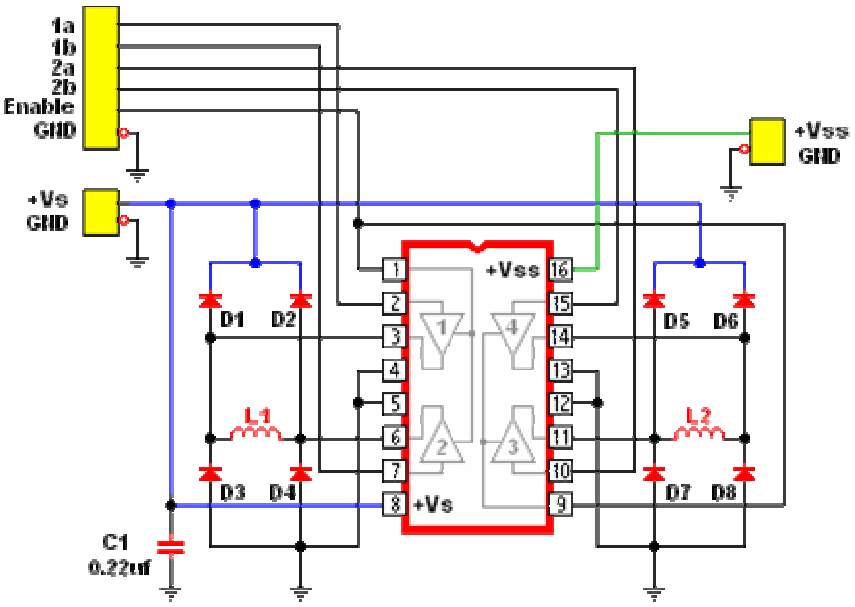
El tema es que para hacerlo debes invertir las polaridades de los terminales de las bobinas 1 y 2 en una determinada secuencia para lograr un giro a derecha, y en secuencia opuesta para que gire a izquierda, la secuencia sería la que se muestra en esta tabla...

Nº de Pasos	1a	1b	2a	2b
Paso 1	+Vcc	GND	+Vcc	GND
Paso 2	+Vcc	GND	GND	+Vcc
Paso 3	GND	+Vcc	GND	+Vcc
Paso 4	GND	+Vcc	+Vcc	GND

Recuerda que 1a y 1b corresponden a un misma bobina, mientras 2a y 2b corresponden a la otra...

Esto de invertir polaridades ya lo vimos anteriormente, lo que necesitamos ahora es la interfaz para controlar estos motores, ya que en la mayoría de los casos se hace a través de un microcontrolador, o por medio de la PC y como estos entregan muy poca corriente nos la tenemos que arreglar.

Una de las mejores opciones para controlar estos motores es hacer uso del Driver L293B que ya lo mencionamos anteriormente, el circuito en cuestión sería el siguiente...



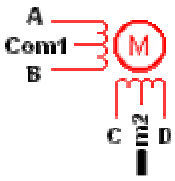
En el esquema **L1** y **L2** son las bobinas del motor, los diodos D1 a D8 son para proteger al integrado de las sobre tensiones generadas por dichas bobinas, las líneas marcadas en azul corresponden a la tensión de alimentación de los motores, mientras la marcada en verde a los niveles TTL de control del integrado, los terminales 1 y 9 se unieron para hacer un solo terminal de habilitación, y finalmente 1a, 1b, 2a y 2b son las entradas de control para la secuencia de las bobinas del motor, este circuito puede servir de base para muchos proyectos, ya sea controlado por PC o por microcontrolador.

De ahora en más todo se reduce a seleccionar un lenguaje de programación y empezar a codificar las secuencias según la tabla anterior.

Ahora vamos por el que sigue...

Motores Unipolares

Estos motores comparado a los anteriores tienen sus ventajas, a pesar de tener mas de 4 cables son más fáciles de controlar, esto se debe a que tienen un terminal común a ambas bobinas. Una forma de identificar cada uno de los cables es analizar la forma de conexión interna de estos motores.



En la imagen los dos bobinados del motor se encuentran separados, pero ambos tienen un terminal central el cual lo llamaremos común (Com1, Com2) jeje, esto parece de puertos... :o))

Bueno, en fin, la cuestión es que este motor tiene 6 cables, y ahora vamos a ponerle nombre a cada uno de ellos. Con el multímetro en modo ohmetro comenzamos a medir resistencias por todos los cables y para mayor sorpresa solo se obtienen tres valores distintos y que se repiten varias veces...

- No marca nada
- 47 ohm
- 100 ohm

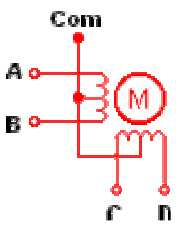
Eso me dio a mi, pero analicémoslo un poco...

100 Ohm es el mayor valor por lo tanto corresponde a los extremos de las bobinas, es decir A-B o bien C-D.

47 Ohm es aproximadamente la mitad de 100, por tanto esa debe ser la resistencia entre el terminal común y ambos extremos de una bobina, por ejemplo entre A-Com1 o B-Com1, o bien en la otra bobina, C-Com2 o D-Com2.

Lo que queda pendiente es cuando no marca nada, y bueno es que en ese momento se midieron los cables de bobinas distintas.

Ahora suponte que unes los terminales Com1 y Com2, entonces te quedas con un motor de 5 cables.



Aquí la resistencia entre cualquier terminal y el común es la misma y aproximadamente la mitad de la resistencia entre los extremos de las bobinas.

Eso fue para identificar el cable común, ahora vamos por los otros...

Como unimos los cables comunes de cada bobina los cuatro cables restantes serán A, B, C y D, y esto ya es a lo guapo, conecta el terminal común al positivo de la fuente de alimentación, toma uno de los 4 cables que te quedaron, lo bautizas como **A** y lo mandas a GND y no lo sacas de ahí hasta que te lo diga, el motor quedará enclavado en una sola posición, ahora abre los ojos bien grandes y sostén otro de los tres que te quedaron, presta mucha atención en esta oportunidad ya que cuando lo conectes a GND el motor dará un primer paso *y luego ya le estaremos enseñando a caminar jaja...!!!*

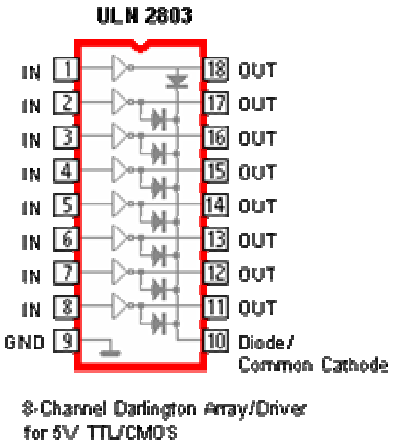
Aquí pueden ocurrir 3 cosas

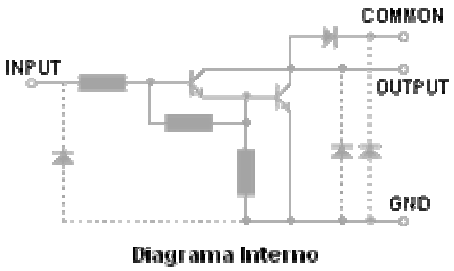
- Que el motor gire a derecha, lo bautizas como B
- Que gire a izquierda, lo nombras D
- Si no pasa nada es C

Si este último cable era B entonces lo desconectas y manteniendo A Conectado buscas D, es decir que gire a izquierda y bueno, C es el que quedó libre. ahora si ya estamos listos para comenzar.

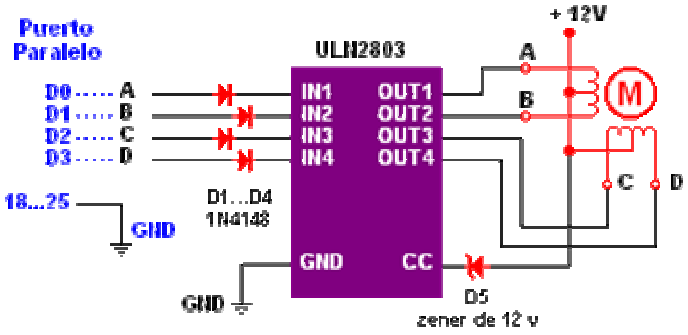
Un circuito para ponerlo a prueba

Lo que necesitamos ahora es un circuito para manejar este motor, y lo vamos a hacer fácil, mira, te consigues un ULN2003 o un ULN2803, que creo es 4 ó 5 veces más económico que el L293B, y es algo así...





Se trata de un array de transistores Dárlington capaz de manejar hasta 500mA en sus salidas, sólo debes tener en cuenta que las salidas están invertidas respecto de las entradas, observa el diagrama interno de una de ellas, se puede apreciar que son de colector abierto. Bien, ahora montemos el siguiente circuito...



Y sí..., lo pondremos a funcionar con el puerto paralelo, no es que sea miedoso, pero por si las moscas, le puse 4 diodos 1N4148, y a demás un diodo zener de 12v para proteger al integrado, este circuito lo puedes montar en una placa de pruebas, que creo está demás decirlo.

Nos queda saber como será la secuencia para poner en marcha estos motores.

Antes de comenzar a explicarlo observa un momento el esquema del circuito y podrás darte cuenta, que los bobinados del motor requieren un pulso de señal negativa para ser activados, como el ULN tiene sus salidas invertidas, cada vez que envíes un "1" por el pin **IN_n** se transformará en "0" a la salida, es decir en el pin **OUT** correspondiente.

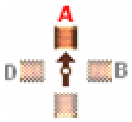
Bien, ahora sí pasemos a lo que sigue...

Secuencia para Motores-PaP Unipolares

Estas secuencias siguen el orden indicado en cada tabla para hacer que el motor gire en un sentido, si se desea que gire en sentido opuesto, sólo se debe invertir dicha secuencia.

Básicamente alcancé a conocer 3 formas de controlar estos motores-PaP, la primera es realizar una secuencia que activa una bobina por pulso...

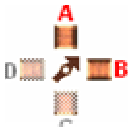
Nº de Pasos	A	B	C	D
Paso 1	1	0	0	0
Paso 2	0	1	0	0
Paso 3	0	0	1	0
Paso 4	0	0	0	1



La verdad es que con esto bastaría, pero al trabajar con una sola bobina se pierde un poco el torque del motor.

La otra propuesta es activar las bobinas de a dos, en este caso el campo magnético se duplica, y en consecuencia el motor tiene mayor fuerza de giro y retención, la secuencia sería la siguiente...

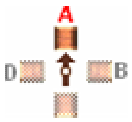
Nº de Pasos	A	B	C	D
Paso 1	1	1	0	0
Paso 2	0	1	1	0
Paso 3	0	0	1	1
Paso 4	1	0	0	1



Lo que noté con este tipo de secuencia es que los movimientos resultan demasiado bruscos, y encima las baterías no te aguantan mucho tiempo, pero dicen que este es de los más recomendados.

En estas dos formas de control vistas anteriormente se respetan la cantidad de pasos preestablecidas para cada motor, por ej., si tiene un ángulo de giro de 90º, con 4 pasos das una vuelta completa, pero también puedes hacerlo con 8 pasos, para lo cual deberías programar la siguiente secuencia.

Nº de Pasos	A	B	C	D
Paso 1	1	0	0	0
Paso 2	1	1	0	0
Paso 3	0	1	0	0
Paso 4	0	1	1	0
Paso 5	0	0	1	0
Paso 6	0	0	1	1
Paso 7	0	0	0	1
Paso 8	1	0	0	1



Esta es una combinación de las dos anteriores, y conocida como secuencia de medio paso, por si te preguntas que pasa con la fuerza de giro, sólo puedo decirte que en este caso es bueno tener en cuenta la inercia del motor cuando este se encuentra en movimiento.

También deberías saber que hay un tiempo determinado para realizar la secuencia en cada uno de los pasos que se debe dar, ya que si la velocidad de los pulsos es demasiado alta, es posible que el motor se vuelva loco y gire en el sentido que se le ocurra, o bien quedarse titubeando en una sola posición sin saber que demonios hacer, respecto a eso, no queda otra que probar, y ver que pasa...

Lo más interesante esta por venir...

Software de Prueba para el control de Motores PaP

De todo lo visto anteriormente no hay nada mejor que ponerlo en práctica, así es que me tomé el trabajo de hacer un pequeño programita en VBasic 6, para ver el funcionamiento de este motor, observa...

jeje, no es eeeeeeel programa, pero sirve para iniciarnos en el tema. Antes de que comiences a preguntar, te diré que lo probé con Windows'98, Windows Millennium, y Windows'XP y funciona de 10, debo aclarar que en Windows'XP tuve un par de inconvenientes al utilizar inpout32.dll, así que la dejé, en su lugar conseguí una nueva librería y su nombre es **IO.dll**, toda la información que necesites de esta librería, la puedes conseguir en http://www.geekhideout.com/iodll.shtml, y es de los mismos creadores de **Parallel Port Monitor**, lo que tiene de bueno, es el soporte para las distintas versiones de Windows y puedes usarla sin tener que especificar en cual trabajas, que bueno no...!!!, claro queeee está en inglés...

Otra cosa que quiero mencionar, parte del código lo saqué de una de las ediciones de la USERS más precisamente la Extra#45, así es que te puede parecer familiar el código fuente de este programita, y como siempre, no puedo estar tranquilo si no le modifico algo, en fin, ahí está...

Todos los detalles de la programación se encuentran en los comentarios del código fuente, el cual puedes bajarlo directamente desde los enlaces al final de este libro.

Por cierto, también incluye la dll y el ejecutable, *por si las moscas...!!!*

Por cierto, vuelvo a repetir...

Cualquier daño ocasionado a tu ordenador por el uso de esta información, no será mi responsabilidad, de acuerdo...???

Ten mucha precaución cuando trabajes con los puertos de tu máquina, ya que cualquier error puede traerte grandes dolores de cabeza...!!!

Y…. dinero !!!

Información técnica del L293B

L293, datos del fabricante

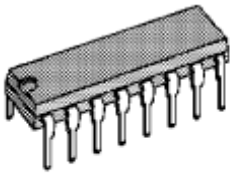
DRIVER PUSH-PULL DE 4 CANALES

Hoja de características con permiso de SGS-Thomson.

SGS-THOMSON MICROELECTRONICS

L293B

- Corriente de salida de 1 A por canal.
- Corriente de salida de pico de 2 A por canal.
- Señal para la habilitación de las salidas.
- Alta inmunidad al ruido.
- Alimentación para las cargas separada de la alimentación de control.
- Protección contra sobre - temperaturas.



DIP16

ORDERING NUMBER : L293B

DESCRIPCIÓN

El L293B es un driver de 4 canales capaz de proporcionar una corriente de salida de hasta 1A por canal. Cada canal es controlado por señales de entrada compatibles TTL y cada pareja de canales dispone de una señal de habilitación que desconecta las salidas de los mismos.

Dispone de una patilla para la alimentación de las cargas que se están controlando, de forma que dicha alimentación es independiente de la lógica de control.

La Figura 2 muestra el encapsulado de 16 pines, la distribución de patillas y la descripción de las mismas.

Pin	Nombre	Descripción
1	Chip Enable 1	Habilitación de los canales 1 y 2
2	Input 1	Entrada del Canal 1
3	Output 1	Salida del Canal 1
4	GND	Tierra de Alimentación
5	GND	Tierra de Alimentación
6	Output 2	Salida del Canal 2
7	Input 2	Entrada del Canal 1
8	Vs	Alimentación de las cargas
9	Chip Enable 2	Habilitación de los canales 3 y 4
10	Input 3	Entrada del Canal 3
11	Output 3	Salida del Canal 3
12	GND	Tierra de Alimentación
13	GND	Tierra de Alimentación
14	Output 4	Salida del Canal 4
15	Input 4	Entrada del Canal 4

Patillaje

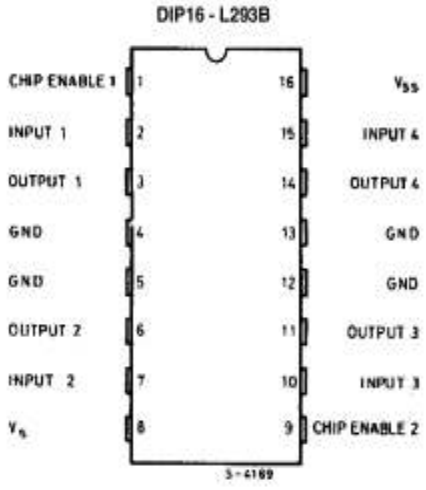


Figura 2.- Descripción de los Pines del L293B

DIAGRAMA DE BLOQUES

En la Figura 3, se muestra el diagrama de bloques del L293B. La señal de control **EN1** activa la pareja de canales formada por los drivers 1 y 2. La señal **EN2** activa la pareja de drivers 3 y 4. Las salidas **OUTN** se asocian con las correspondientes **OUTn**. Las señales de salida son amplificadas respecto a las de entrada tanto en tensión (hasta +Vss) como en corriente (máx. 1 A).

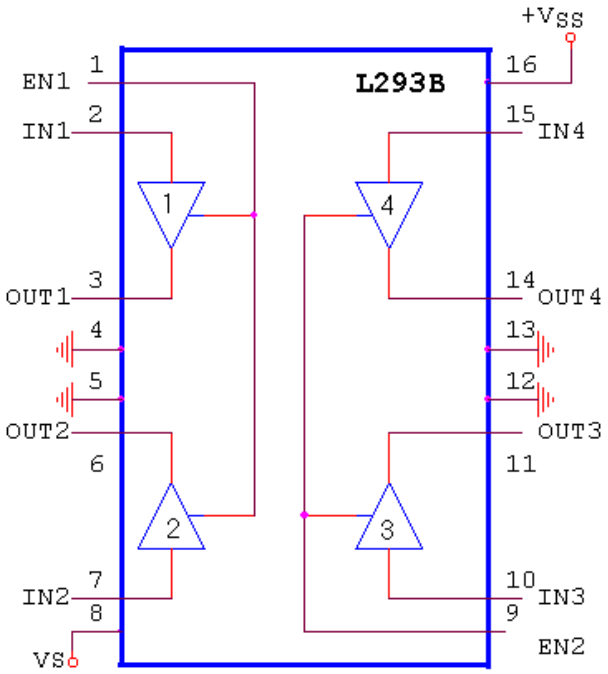


Figura 3.- Diagrama de bloques del L293B

La tabla de funcionamiento para cada uno de los driver es la siguiente:

V_{INn}	V_{OUTn}	V_{ENn}
H	H	H
L	L	H
H	Z	L
L	Z	L

Donde:

H = Nivel alto "1"
L = Nivel bajo "0"
Z = Alta Impedancia

PARAMETROS

RANGOS ABSOLUTOS MAXIMOS

Símbolo	Parámetro	Valor	Unidades
V_s	Tensión de alimentación para las cargas	36	V
V_{ss}	Tensión de alimentación de la lógica	36	V
V_i	Tensión de entrada	7	V
V_{inh}	Tensión de habilitación	7	V
I_{out}	Intensidad de pico de salida	2	A
P_{tot}	Potencia total de disipación	5	W
T_{stg}, T_j	Temperatura de almacenaje y de la unión	-40 a +150	ºC

CARÁCTERISTICAS ELECTRICAS
Para cada canal, Vs = 24V, Vss = 5V, Tamb =25ºC

Símbolo	Parámetro	Condiciones de Test	Min	Tipica	Max	Unidades
V_S	Tensión de alimentación de las cargas		V_{SS}		36	V
V_{SS}	Tensión de alimentación de la lógica		4.5		36	V
I_S	Corriente total de reposo	V_i = L I_O=0 V_{inh} =H V_i = H I_O=0 V_{inh} =H V_{inh} =L		2 16	6 24 4	mA
I_{SS}	Corriente total de reposo con señal de control	V_i = L I_O=0 V_{inh} =H V_i = H I_O=0 V_{inh} =H V_{inh} =L		44 16 16	60 22 24	mA
V_{IL}	Tensión de entrada a nivel bajo		-0.3		1.5	V
V_{IH}	Tensión de entrada a nivel alto	$V_{SS} \leq 7V$ $V_{SS} > 7V$	2.3 2.3		V_{SS} 7	V
I_{IL}	Corriente de entrada a nivel bajo	V_{iL} =1.5V			-10	µA
I_{IH}	Corriente de entrada a nivel alto	$2.3V \leq V_{IH} \leq V_{SS} - 0.6$		30	100	µA
V_{inhL}	Tensión de habilitación a nivel bajo		-0,3		1.5	V
V_{inhH}	Tensión de habilitación a nivel alto	$V_{SS} \leq 7V$ $V_{SS} > 7V$	2.3 2.3		V_{SS} 7	V
I_{inhL}	Corriente de habilitación a nivel bajo	V_{InhL} = 1.5V		-30V	-100	µA
I_{inhH}	Corriente de habilitación a nivel alto	$2.3V \leq V_{IH} \leq V_{SS} - 0.6$			± 10	µA
$V_{CEsatH.}$	Tensión de salida con la fuente saturada	I_O = 1 A		1.4	1.8	V
$V_{CesatL.}$	Tensión de salida con el sumidero saturado	I_O = 1 A		1.2	1.8	V

APLICACIONES.
En este apartado se muestran distintas configuraciones de conexión de motores al L293B.

1. GIRO DE 2 MOTORES EN UNICO SENTIDO.
En la Figura 4 se muestra el modo de funcionamiento de dos motores de corriente continua que giran en un único sentido.

- El motor M1 se activa al poner a nivel bajo la entrada de control A.
- El motor M2 se activa al poner a nivel alto la entrada de control B.

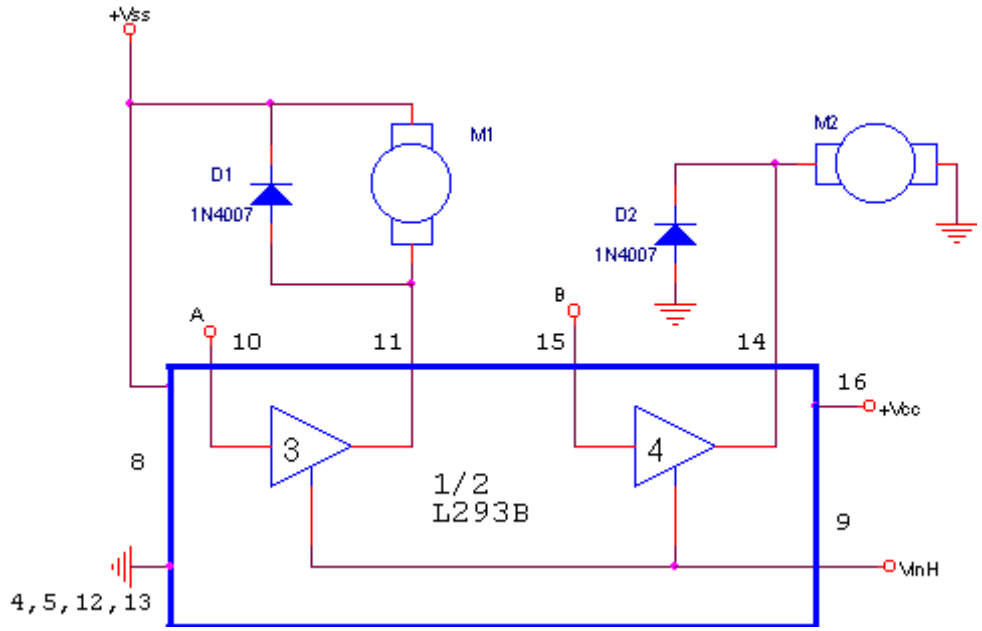


Figura 4.- Conexión de 2 motores de continua M1 activado por “0” y M2 por “1”

Su tabla de funcionamiento es la siguiente:

V_{inh}	A	M1	B	M2
H	H	Parada rápida del motor	H	Giro
H	L	Giro	L	Parada rápida del motor
L	X	Motor desconectado, giro libre	X	Motor desconectado, giro libre

Tabla de verdad del circuito de la Figura 4

Los diodos D1 y D2, están conectados para proteger el circuito cuando se generan los picos de arranque de los motores. Si no se trabaja a máxima potencia de trabajo pueden eliminarse del circuito.

2. CONTROL DEL GIRO DE UN MOTOR EN LOS DOS SENTIDOS

El circuito de la Figura 5 permite controlar el doble sentido de giro del motor. Cuando la entrada **C** está a nivel bajo y la **D** a nivel alto, el motor gira hacia la izquieda. Cambiando la entrada **C** a nivel alto y la **D** a nivel Bajo, se cambia el sentido de giro del motor hacia la derecha.

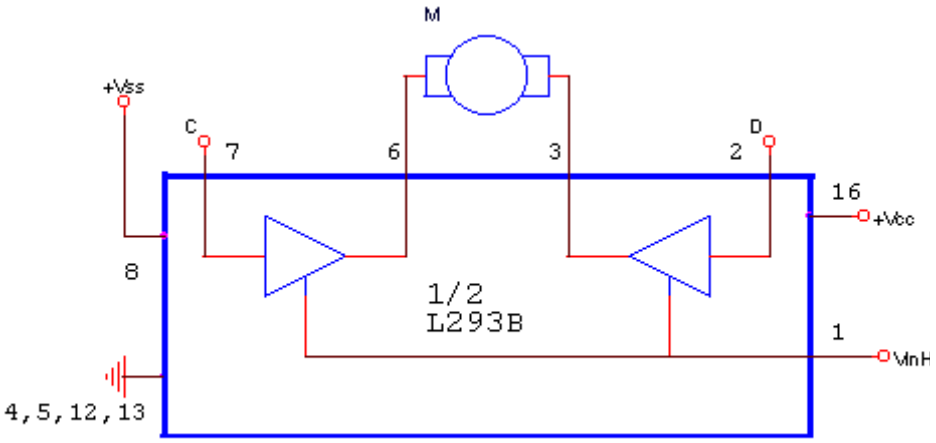


Figura 5.- Circuito de control para el doble giro de un motor de corriente continua

Si se quiere proteger el circuito contra posibles picos de corriente inversa cuando se arranca el motor, se recomienda conectar unos diodos tal y como se muestra en la Figura 6.

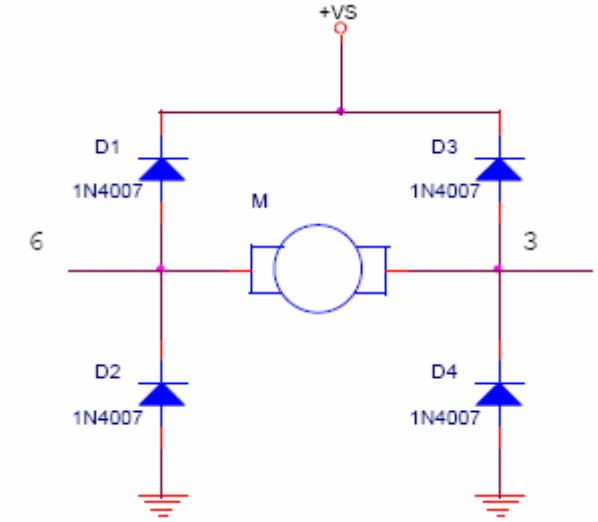


Figura 6.- Circuito de protección para el L293 para evitar sobre corrientes inversas al arrancar el motor.

En este caso la tabla de funcionamiento es la siguiente:

V_{inh}	A	B	M
H	L	L	Parada rápida del motor
H	H	H	Parada rápida del motor
H	L	H	Giro a la Izquierda
H	H	L	Giro a la derecha
L	X	X	Motor desconectado, giro libre

3. CONTROL DE UN MOTOR PASO A PASO BIPOLAR

En la Figura 7 se muestra una forma de conectar un motor bipolar paso a paso. En este caso habrá que generar la secuencia adecuada al motor paso a paso para poder excitar de forma correcta sus bobinas.

La forma de proteger el circuito contra las corrientes que se producen en el momento de arranque del motor sería el mismo que el de la Figura 6 pero para cada una de las bobinas del motor paso a paso, es decir, utilizando 8 diodos.

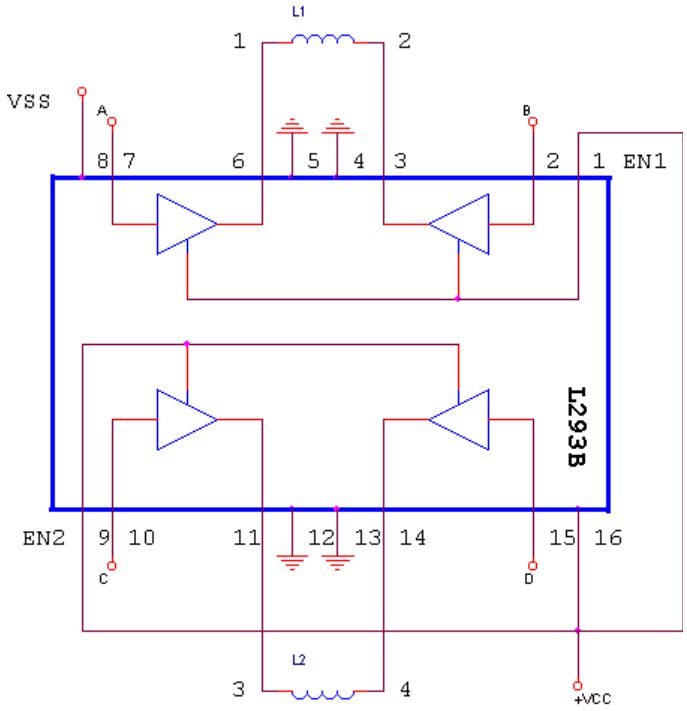


Figura 7.- Conexión de un motor paso a paso bipolar al L293B.

La secuencia a aplicar a las bobinas del motor paso a paso, depende del tipo de motor bipolar a utilizar, pero el fabricante suele indica con una tabla como la siguiente o un cronograma la secuencia a aplicar para que el motor gire en un sentido u otro.

Forma e ataque a las bobinas				
Sentido de giro horario				
PASO	L1-1	L1-2	L2-3	L2-4
1	+	-	+	-
2	-	+	+	-
3	-	+	-	+
4	+	-	-	+

Forma e ataque a las bobinas				
Sentido de giro anti - horario				
Paso	L1-1	L1-2	L2-3	L2-4
1	+	-	-	+
2	-	+	-	+
3	-	+	+	-
4	+	-	+	-

CIRCUITO DE APLICACIÓN.

Para realizar las prácticas con motores y el L293b implementaremos el circuito de la Figura 8, que nos permitirá controlar de forma sencilla y tanto el control de giro de dos motores de corriente continua o un solo motor paso a paso de 4 o 6 hilos.

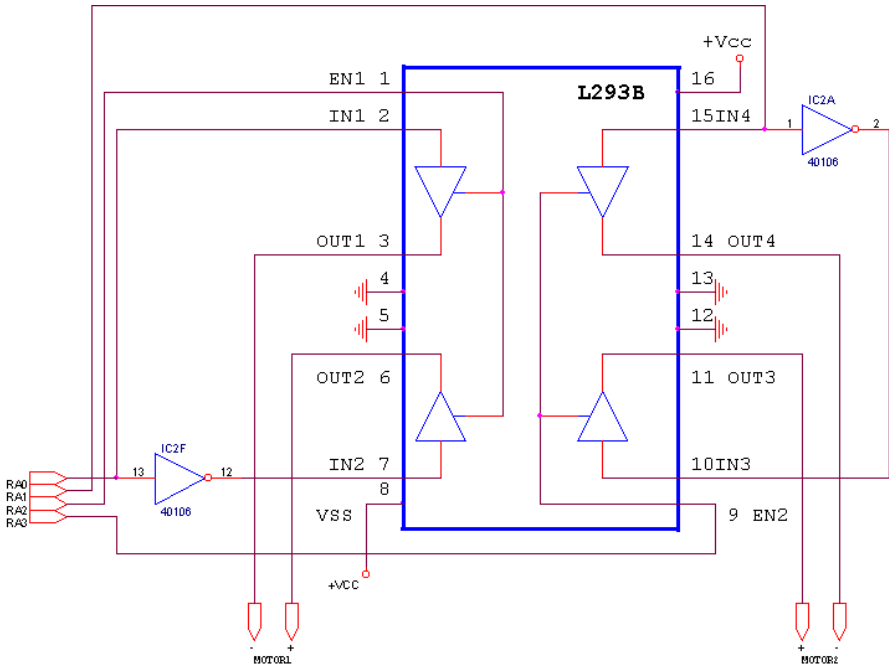


Figura 8.- Circuito de control para motores DC y paso a paso

Elctrónica digital

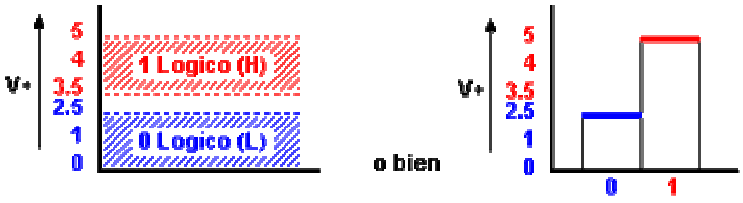
Qué es Electrónica Digital?

Obviamente es una ciencia que estudia las señales eléctricas, pero en este caso son señales discretas, es decir, están bien identificadas, razón por la cual a un determinado nivel de tensión se lo llama estado alto (**H**igh) o Uno lógico; y a otro, estado bajo (**L**ow) o Cero lógico.

Suponte que las señales eléctricas con que trabaja un sistema digital son 0V y 5V. Es obvio que 5V será el estado alto o uno lógico, pero bueno, habrá que tener en cuenta que existe la **Lógica Positiva** y la **Lógica Negativa**, veamos cada una de ellas.

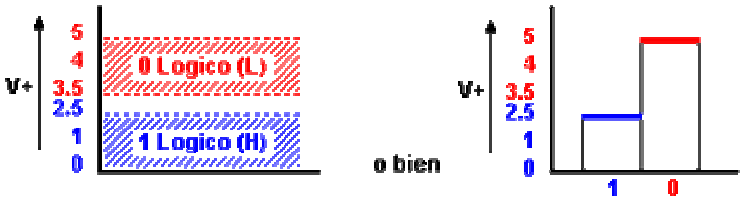
Lógica Positiva

En esta notación al 1 lógico le corresponde el nivel más alto de tensión (positivo, si quieres llamarlo así) y al 0 lógico el nivel mas bajo (que bien podría ser negativo), pero que ocurre cuando la señal no está bien definida...?. Entonces habrá que conocer cuales son los límites para cada tipo de señal (conocido como tensión de histéresis), en este gráfico se puede ver con mayor claridad cada estado lógico y su nivel de tensión.



Lógica Negativa

Aquí ocurre todo lo contrario, es decir, se representa al estado **"1"** con los niveles más bajos de tensión y al **"0"** con los niveles más altos.



Por lo general se suele trabajar con lógica positiva, y así lo haremos en este tutorial, la forma más sencilla de representar estos estados es como se puede ver en el siguiente gráfico.

De ahora en adelante ya sabrás a que nos referimos con estados lógicos **1** y **0**, de todos modos no viene nada mal saber un poco más... ;-)

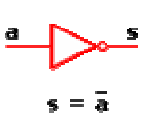
Compuertas Lógicas

Las compuertas lógicas son dispositivos que operan con aquellos estados lógicos mencionados en la página anterior y funcionan igual que una calculadora, de un lado ingresas los datos, ésta realiza una operación, y finalmente, te muestra el resultado.

Cada una de las compuertas lógicas se las representa mediante un **Símbolo**, y la operación que realiza (**Operación lógica**) se corresponde con una tabla, llamada **Tabla de Verdad**, vamos con la primera...

Compuerta NOT

Se trata de un inversor, es decir, invierte el dato de entrada, por ejemplo; si pones su entrada a 1 (nivel alto) obtendrás en su salida un 0 (o nivel bajo), y viceversa. Esta compuerta dispone de una sola entrada. Su operación lógica es **s** igual a **a invertida**



a	s
0	1
1	0

Compuerta AND

Una compuerta AND tiene dos entradas como mínimo y su operación lógica es un producto entre ambas, no es un producto aritmético, aunque en este caso coincidan.

Observa que su salida será alta si sus dos entradas están a nivel alto

a	b	s
0	0	0
0	1	0
1	0	0
1	1	1

Compuerta OR

Al igual que la anterior posee dos entradas como mínimo y la operación lógica, será una suma entre ambas... *Bueno, todo va bien hasta que 1 + 1 = 1*, el tema es que se trata de una puerta **O Inclusiva** es como **a y/o b.**

*Es decir, basta que una de ellas sea **1** para que su salida sea también 1*

a	b	s
0	0	0
0	1	1
1	0	1
1	1	1

Compuerta OR-EX o XOR

Es OR EXclusiva en este caso con dos entradas (puede tener mas, claro...!) y lo que hará con ellas será una suma lógica entre **a** por **b invertida** y **a invertida** por **b**.

Al ser **O Exclusiva su salida será **1** si una y **sólo una** de sus entradas es 1**

$s = a.\bar{b} + \bar{a}.b$

a	b	s
0	0	0
0	1	1
1	0	1
1	1	0

Estas serían básicamente las compuertas mas sencillas. Es momento de complicar esto un poco más...

Compuertas Lógicas Combinadas

Al agregar una compuerta NOT a cada una de las compuertas anteriores, los resultados de sus respectivas tablas de verdad se invierten, y dan origen a tres nuevas compuertas llamadas **N**AND, **N**OR y **N**OR-EX... Veamos ahora como son y cual es el símbolo que las representa...

Compuerta NAND

Responde a la inversión del producto lógico de sus entradas, en su representación simbólica se reemplaza la compuerta NOT por un círculo a la salida de la compuerta AND.

$s = \overline{a \cdot b}$

a	b	s
0	0	1
0	1	1
1	0	1
1	1	0

Compuerta NOR

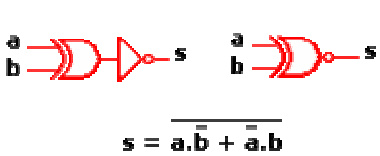
El resultado que se obtiene a la salida de esta compuerta resulta de la inversión de la operación lógica **o inclusiva** es como un **no a y/o b**. Igual que antes, solo agregas un círculo a la compuerta OR y ya tienes una NOR.

$s = \overline{a + b}$

a	b	s
0	0	1
0	1	0
1	0	0
1	1	0

Compuerta NOR-EX

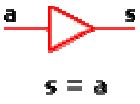
Es simplemente la inversión de la compuerta OR-EX, los resultados se pueden apreciar en la tabla de verdad, que bien podrías compararla con la anterior y notar la diferencia, el símbolo que la representa lo tienes en el siguiente gráfico.



a	b	s
0	0	1
0	1	0
1	0	0
1	1	1

Buffer's

Ya la estaba dejando de lado..., no se si viene bien incluirla aquí pero de todos modos es bueno que la conozcas, en realidad no realiza ninguna operación lógica, su finalidad es amplificar un poco la señal (o refrescarla si se puede decir). Como puedes ver en el siguiente gráfico, la señal de salida es la misma que de entrada.



a	s
0	0
1	1

Hasta aquí de teoría, nos interesa más saber como se hacen evidente estos estados en la práctica, y en qué circuitos integrados se las puede encontrar y más adelante veremos unas cuantas leyes que se pueden aplicar a estas compuertas para obtener los resultados que deseas...

Circuitos Integrados y Circuito de Prueba

Existen varias familias de Circuitos integrados, pero sólo mencionaré dos, los más comunes, que son los TTL y CMOS:

Estos Integrados los puedes caracterizar por el número que corresponde a cada familia según su composición. Por ejemplo;

Los TTL se corresponden con la serie 5400, 7400, 74LSXX, 74HCXX, 74HCTXX etc. algunos 3000 y 9000.

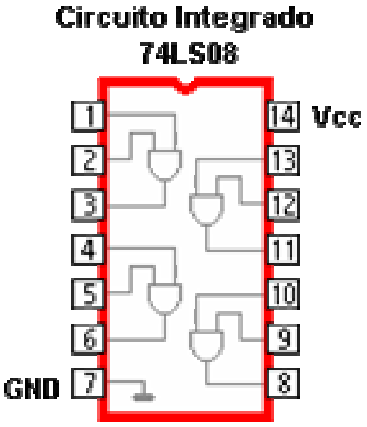
Los C-MOS y MOS se corresponde con la serie CD4000, CD4500, MC14000, 54C00 ó 74C00. en fin...

La pregunta de rigor... Cual es la diferencia entre uno y otro...?, veamos... yo comencé con los C-MOS, ya que disponía del manual de estos integrados, lo bueno es que el máximo nivel de tensión soportado llega en algunos casos a +15V, (especial para torpes...!!!), mientras que para los TTL el nivel superior de tensión alcanza en algunos casos a los +12V aproximadamente, pero claro estos son límites extremos, lo común en estos últimos es utilizar +5V y así son felices.

Otra característica es la velocidad de transmisión de datos, resulta ser, que los circuitos TTL son mas rápidos que los C-MOS, por eso su mayor uso en sistemas de computación.

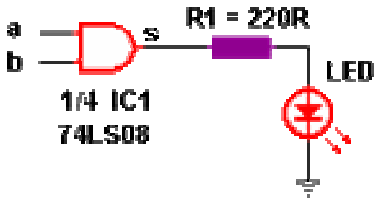
Suficiente... de todos modos es importante que busques la hoja de datos o datasheet del integrado en cuestión, distribuido de forma gratuita por cada fabricante y disponible en Internet... *donde más...?*

Veamos lo que encontramos en uno de ellos; en este caso un Circuito integrado 74LS08, un TTL, es una cuádruple compuerta AND. Es importante que notes el sentido en que están numerados los pines y esto es general, para todo tipo de integrado...

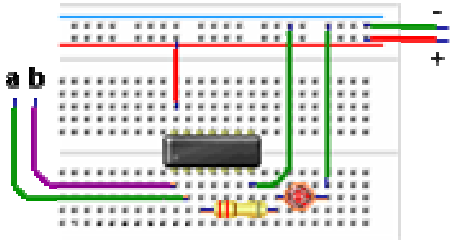


Comenzaremos con este integrado para verificar el comportamiento de las compuertas vistas anteriormente. El representado en el gráfico marca una de las compuertas que será puesta a prueba, para ello utilizaremos un fuente regulada de +5V, un LED una resistencia de 220 ohm, y por supuesto el IC que corresponda y la placa de prueba.

El esquema es el siguiente...



En el esquema está marcada la compuerta, como 1 de 4 disponibles en el Integrado 74LS08, los extremos **a** y **b** son las entradas que deberás llevar a un 1 lógico (+5V) ó 0 lógico (GND), el resultado en la salida **s** de la compuerta se verá reflejado en el LED, LED encendido (1 lógico) y LED apagado (0 lógico), no olvides conectar los terminales de alimentación que en este caso son el pin 7 a GND y el 14 a +5V. Montado en la placa de prueba te quedaría algo así...



Esto es a modo de ejemplo, *Sólo debes reemplazar IC1*, que es el Circuito Integrado que está a prueba para verificar su tabla de verdad.

Un poco de Leyes

Antes de seguir... Lo primero y más importante es que trates de interpretar la forma en que realizan sus operaciones cada compuerta lógica, ya que a partir de ahora las lecciones se complican un poco más. Practica y verifica cada una de las tablas de verdad.

Leyes de De Morgan

Se trata simplemente de una combinación de compuertas, de tal modo de encontrar una equivalencia entre ellas, esto viene a consecuencia de que en algunos casos no dispones del integrado que necesitas, pero si de otros que podrían producir los mismos resultados que estas buscando.

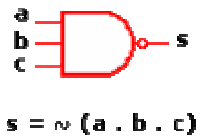
Para interpretar mejor lo que viene, considera a las señales de entrada como variables y al resultado como una función entre ellas. El símbolo de negación (operador NOT) lo representaré por **"~"**, por ejemplo: **a . ~ b** significa **a AND NOTb**, se entendió...?

1º Ley:

El producto lógico negado de varias variables lógicas es igual a la suma lógica de cada una de dichas variables negadas. Si tomamos un ejemplo para 3 variables tendríamos..

$$\sim (a.b.c) = \sim a + \sim b + \sim c$$

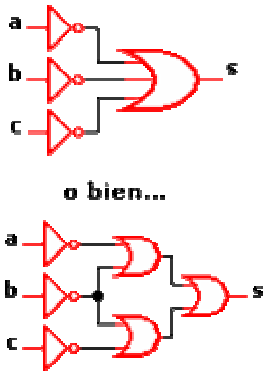
El primer miembro de esta ecuación equivale a una compuerta NAND de 3 entradas, representada en el siguiente gráfico y con su respectiva tabla de verdad.



s = ∾ (a . b . c)

a	b	c	s
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

El segundo miembro de la ecuación se lo puede obtener de dos formas...



a	b	c	s
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

s = ∾a + ∾b + ∾c

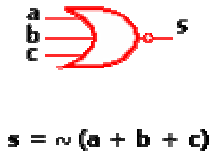
Fíjate que la tabla de verdad es la misma, ya que los resultados obtenidos son iguales. Acabamos de verificar la primera ley.

2º Ley:

La suma lógica negada de varias variables lógicas es igual al producto de cada una de dichas variables negadas...

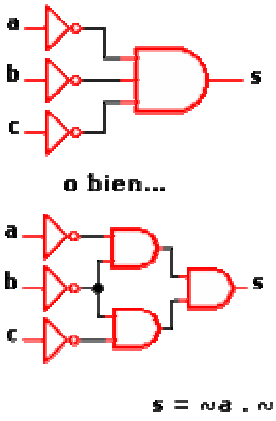
~ (a + b + c) = ~a . ~b . ~c

El primer miembro de esta ecuación equivale a una compuerta NOR de 3 entradas y la representamos con su tabla de verdad...



a	b	c	s
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

El segundo miembro de la ecuación se lo puede obtener de diferentes forma, aquí cité solo dos...



a	b	c	s
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Nuevamente... Observa que la tabla de verdad es la misma que para el primer miembro en el gráfico anterior. Acabamos así de verificar la segunda ley de De Morgan.

Para concluir... Con estas dos leyes puedes llegar a una gran variedad de conclusiones, por ejemplo...

Para obtener una compuerta AND puedes utilizar una compuerta NOR con sus entradas negadas, o sea...

a . b = ~(~a + ~b)

Para obtener una compuerta OR puedes utilizar una compuerta NAND con sus entradas negadas, es decir...

a + b =~(~a . ~b)

Para obtener una compuerta NAND utiliza una compuerta OR con sus dos entradas negadas, como indica la primera ley de De Morgan...

~ (a.b) = ~a + ~b

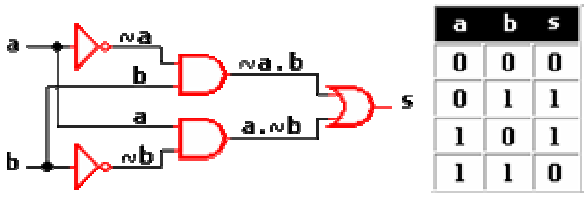
Para obtener una compuerta NOR utiliza una compuerta AND con sus entradas negadas, ...eso dice la 2º ley de De Morgan, así que... habrá que obedecer...

~(a + b) = ~a . ~b

La compuerta OR-EX tiene la particularidad de entregar un nivel alto cuando una y sólo una de sus entradas se encuentra en nivel alto. Si bien su función se puede representar como sigue...

s = a . ~b + ~a . b

te puedes dar cuenta que esta ecuación te indica las compuertas a utilizar, y terminarás en esto...



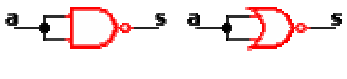
a	b	s
0	0	0
0	1	1
1	0	1
1	1	0

s = a.~b + ~a.b

Para obtener una compuerta NOR-EX agregas una compuerta NOT a la salida de la compuerta OR-EX vista anteriormente y ya la tendrás. Recuerda que su función es...

s = ~(a . ~b + ~a . b)

Para obtener Inversores (NOT) puedes hacer uso de compuertas NOR o compuertas NAND, simplemente uniendo sus entradas.



s=~a

Existen muchas opciones más, pero bueno... ya las irás descubriendo, o las iremos citando a medida que vayan apareciendo, de todos modos valió la pena. *No crees...?*

Más sobre funciones y operadores lógicos

A estas alturas ya estamos muy familiarizados con las funciones de todos los operadores lógicos y sus tablas de verdad, todo vino bien..., pero... qué hago si dispongo de tres entradas (a, b y c) y deseo que los estados altos sólo se den en las combinaciones 0, 2, 4, 5 y 6 (decimal)...? Cómo combino las compuertas...? y lo peor, Qué compuertas utilizo...?. No te preocupes, yo tengo la solución, ...pégate un tiro... :o))

Bueno... NO...!!!, mejor no. Trataré de dar una solución verdadera a tu problema, preparado...?

Mapas de Karnaugh

Podría definirlo como un método para encontrar la forma más sencilla de representar una función lógica.

Esto es... Encontrar la función que relaciona todas las variables disponibles, de tal modo que el resultado sea el que se está buscando.

Para esto vamos a aclarar tres conceptos que son fundamentales

a)- Minitérmino Es cada una de las combinaciones posibles entre todas las variables disponibles, por ejemplo con 2 variables obtienes 4 minitérminos; con 3 obtienes 8; con 4, 16 etc., como te darás cuenta se puede encontrar la cantidad de minitérminos haciendo $\mathbf{2^n}$ donde **n** es el número de variables disponibles.

b)- Numeración de un minitérmino Cada minitérmino es numerado en decimal de acuerdo a la combinación de las variables y su equivalente en binario así...

a	b	Minit.
0	0	Minit. 0
0	1	Minit. 1
1	0	Minit. 2
1	1	Minit. 3

Bien... El Mapa de Karnaugh representa la misma tabla de verdad a través de una matriz, en la cual en la primer fila y la primer columna se indican las posibles combinaciones de las variables. Aquí tienes tres mapas para 2, 3 y 4 variables...

Para 2 Variables

a\b	0	1
0	0	1
1	2	3

Para 3 Variables

a\bc	00	01	11	10
0	0	1	3	2
1	4	5	7	6

Para 4 Variables

ab\cd	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Analicemos el mapa para cuatro variables, las dos primeras columnas (columnas adyacentes) difieren sólo en la variable **d**, y c permanece sin cambio, en la segunda y tercer columna (columnas adyacentes) cambia **c**, y d permanece sin cambio, ocurre lo mismo en las filas. En general se dice que...

Dos columnas o filas adyacentes sólo pueden diferir en el estado de una de sus variables

Observa también que según lo dicho anteriormente la primer columna con la última serían adyacentes, al igual que la primer fila y la última, ya que sólo difieren en una de sus variables

c)- Valor lógico de un minitérmino (esos que estaban escritos en rojo), bien, estos deben tener un valor lógico, y es el que resulta de la operación que se realiza entre las variables. lógicamente **0** ó **1**

Listo... Lo que haremos ahora será colocar el valor de cada minitérmino según la tabla de verdad que estamos buscando... diablos...!!! en este momento no se me ocurre nada, bueno si, trabajemos con esta...

a\bc	00	01	11	10
0	1	1	0	0
1	1	0	0	1

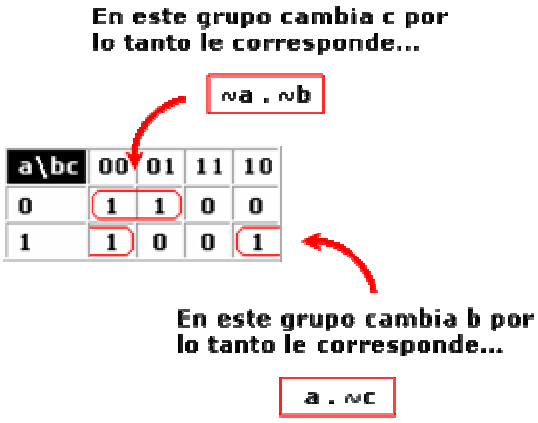
a	b	c	s
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

El siguiente paso, es agrupar los unos adyacentes (horizontal o verticalmente) en grupos de potencias de 2, es decir, en grupos de 2, de 4, de 8 etc... y nos quedaría así...

a\bc	00	01	11	10
0	1	1	0	0
1	1	0	0	1

Te preguntarás que pasó con la fila de abajo... bueno, es porque no estas atento...!!! Recuerda que la primer columna y la última son adyacentes, por lo tanto sus minitérminos también lo son.

De ahora en más a cada grupo de unos se le asigna la unión (producto lógico) de las variables que se mantienen constante (ya sea uno o cero) ignorando aquellas que cambian, tal como se puede ver en esta imagen...



Para terminar, simplemente se realiza la suma lógica entre los términos obtenidos dando como resultado la función que estamos buscando, es decir...

f = (~a . ~b) + (a . ~c)

Puedes plantear tu problema como una función de variables, en nuestro ejemplo quedaría de esta forma...

f(a, b, c) = S(0, 1, 4, 6)

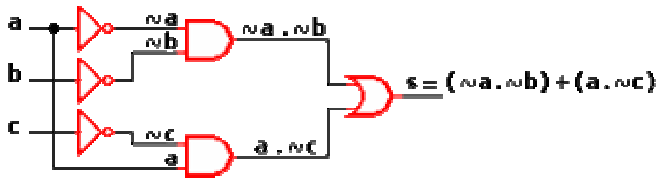
F es la función buscada
(a, b, c) son las variables utilizadas
(0, 1, 4, 6) son los minitérminos que dan como resultado **1** o un nivel alto.
S La sumatoria de las funciones que producen el estado alto en dichos minitérminos.

Sólo resta convertir esa función en su circuito eléctrico correspondiente. Veamos, si la función es...

f = (~a . ~b) + (a . ~c) o sea...

(NOT a AND NOT b) OR (a AND NOT c)

El esquema eléctrico que le corresponde es el que viene a continuación...



El resultado de todo este lío, es un circuito con la menor cantidad de compuertas posibles, lo cual lo hace más económico, por otro lado cumple totalmente con la tabla de verdad planteada al inicio del problema, y a demás recuerda que al tener menor cantidad de compuertas la transmisión de datos se hace más rápida.

En fin... Solucionado el problema...!!!

Por cierto, un día, mientras merodeaba por la red me encontré con un pequeño programa que hace todo este trabajo por su cuenta, El programa se llama **Karma** Creado por Pablo Fernández Fraga, mis saludos Pablo...!!! está muy, pero muy bueno...!!!

Basta por hoy, muy pronto utilizaremos toda esta teoría y el programa de pablo (Karma) para diseñar una tarjeta controladora de motores paso a paso, mientras tanto averigua como funcionan estos motores.

Circuitos Astables

Osciladores, Multivibradores o Circuitos Astables - Parte I

Esta vez prepara tu placa de pruebas y unos cuantos integrados, que lo que viene será todo práctica... Seguramente te llama la atención el nombre **Astable**. Bien... lo explicaré brevemente...

Existen tres circuitos clasificados según la forma en que retienen o memorizan el estado que adoptan sus salidas, estos son...

Circuitos Biestables o Flip-Flop (FF):

Son aquellos que cambian de estado cada vez que reciben una señal de entrada (ya sea nivel bajo o alto), es decir retienen el dato de salida aunque desaparezca el de entrada. **Conclusión: Poseen dos estados estables**

Circuitos Monoestables:

Estos circuitos cambian de estado sólo si se mantiene la señal de entrada (nivel alto o bajo), si ésta se quita, la salida regresa a su estado anterior.

Conclusión: Poseen un sólo estado estable y otro metaestables

Circuitos Astables o Aestables:

Son circuitos gobernados por una red de tiempo R-C (Resistencia-Capacitor) y un circuito de realimentación, a diferencia de los anteriores se puede decir que **no poseen un estado estable sino dos metaestables**

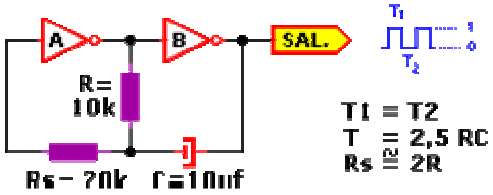
Y a estos últimos nos dedicaremos ahora, los otros dos los trataremos en las próximas lecciones...

De todos los circuitos astables el más conocido es el que se construye con un integrado NE555, el cual ya vimos como hacerlo tiempo atrás (en nuestro tutorial de electrónica básica). La idea es que veas todas las posibilidades que te brindan las compuertas lógicas y ésta es una de ellas, considerando que en muchos circuitos o diseños quedan compuertas libres (sin ser utilizadas) vamos a aprovecharlas para armar circuitos astables, timer's o temporizadores, o yo que se, como le quieras llamar.

Comencemos...

Oscilador Simétrico con compuertas NOT

Fue el primero que se me ocurrió y utiliza dos inversores o compuertas NOT.



Descripción:

Suponte que en determinado momento la salida del inversor B está a nivel "1", entonces su entrada esta a "0", y la entrada del inversor "A" a nivel "1". En esas condiciones **C** se carga a través de **R**, y los inversores permanecen en ese estado.

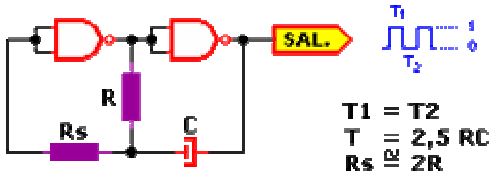
Cuando el capacitor alcanza su carga máxima, se produce la conmutación del inversor "A". Su entrada pasa a "0", su salida a "1" y la salida del inversor "B" a "0", se invierte la polaridad del capacitor y este se descarga, mientras tanto los inversores permanecen sin cambio, una vez descargado, la entrada del inversor "A" pasa nuevamente a "1", y comienza un nuevo ciclo.

Este oscilador es simétrico ya que el tiempo que dura el nivel alto es igual al que permanece en nivel bajo, este tiempo esta dado por **T = 2,5 R C**

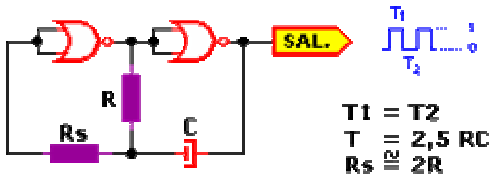
T expresado en segundos
R en Ohms
C en Faradios

Creo yo que fue fácil y sencillo hacerlo, ahora bien, si recordamos aquello de **las leyes de De Morgan** sabrás que uniendo las entradas de compuertas NAND o compuertas NOR obtienes la misma función que los inversores o compuertas NOT, esto me lleva a las siguientes conclusiones...

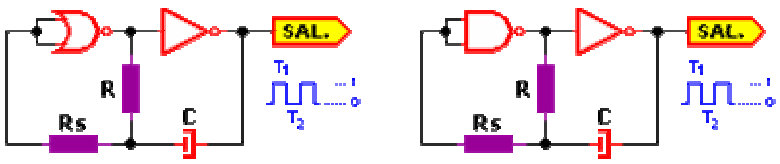
Oscilador Simétrico con compuertas NAND



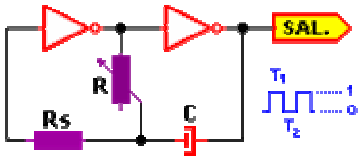
Oscilador Simétrico con compuertas NOR



Como veras, todo se basa en el primero que vimos, y hay más combinaciones todavía..., por ejemplo...



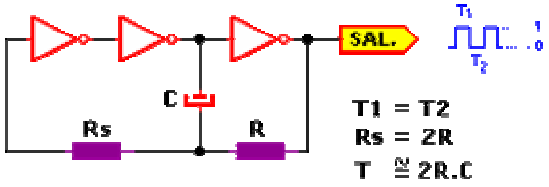
Y así... hasta que me cansé, algo que no mencioné es que puedes controlar la velocidad de estos circuitos, *Cómo...?*, Muy fácil mira...



Aquí **R** es de 100k pero puedes usar otro a ver que ocurre, o cambia el capacitor, bueno, ya verás que hacer... pero sigamos con esto que aquí no termina...

Osciladores, Multivibradores o Circuitos Astables - Parte II

Sabes que probé los osciladores anteriores con un parlante pequeño (*de esos de PC*) pero nada..., hasta que encontré una solución...

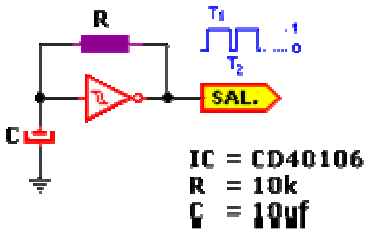


y este sí me dio resultado, hasta puedes reemplazar **R** por un potenciómetro y regular el sonido (es decir, su frecuencia) .

Disparadores Schmitt Trigger

Algo que no vimos hasta ahora son las compuertas SCHMITT TRIGGER o disparadores de Schimitt, son iguales a las compuertas vistas hasta ahora pero tienen la ventaja de tener umbrales de conmutación muy definidos llamados VT+ y VT-, esto hace que puedan reconocer señales que en las compuertas lógicas comunes serían una indeterminación de su estado y llevarlas a estados lógicos definidos, mucho mas definidos que las compuertas comunes que tienen un solo umbral de conmutación.

Se trata de esto...



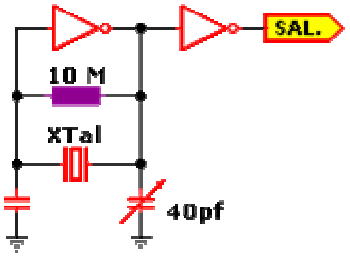
Suponte la salida a nivel lógico **1**, **C** comienza a cargarse a través de **R**, a medida que la tensión crece en la entrada de la compuerta esta alcanza el nivel VT+ y produce la conmutación de la compuerta llevando la salida a nivel **0** y el capacitor comienza su descarga.

Cuando el potencial a la entrada de la compuerta disminuye por debajo del umbral de VT-, se produce nuevamente la conmutación pasando la salida a nivel **1**, y se reinicia el ciclo.

No sólo existen inversores Schmitt Trigger, sino también compuertas AND, OR, NOR, etc, y ya sabes como utilizarlas, pero veamos una posibilidad más de obtener circuitos así...

Oscilador a Cristal

Se trata de un oscilador implementado con dos inversores y un Cristal de cuarzo, el trimer de 40pf se incluye para un ajuste fino de la frecuencia de oscilación, mientras el circuito oscilante en si funciona con un solo inversor, se incluye otro para actuar como etapa separadora. *extraído de un libro de por ahí...* :-P



Hasta aquí..., Te cuento que los Osciladores vistos hasta el momento pueden ser controlados fácilmente, y eso es lo que haremos ahora...

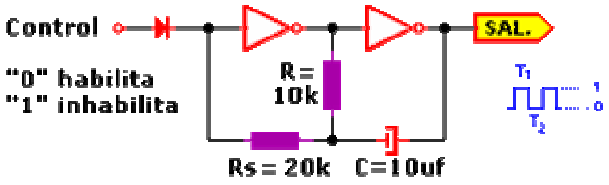
Osciladores, Multivibradores o Circuitos Astables - Parte III

Osciladores Controlados

Se trata simplemente de controlar el momento en que estos deben oscilar. Veamos..., tenemos dos opciones, que sean controlados por un nivel alto o por un nivel bajo.

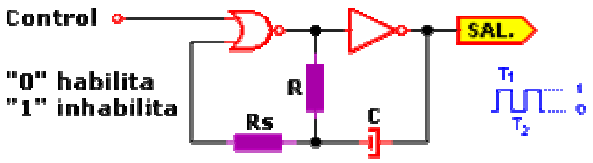
Si tienes en cuenta que los osciladores vistos hasta el momento solo pueden oscilar cambiando el estado de sus entradas en forma alternada, lo que haremos será forzar ese estado a un estado permanente, como dije anteriormente ya sea a **1** o **0**

Vamos al primer ejemplo; lo haremos utilizando un diodo en la entrada del primer inversor, así...

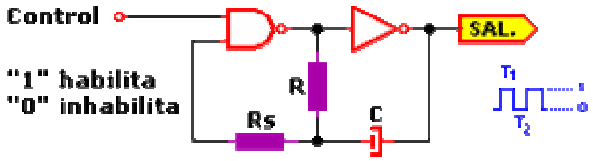


Creo que está claro, si el terminal de control está a nivel **0** el circuito oscilará, si está a nivel **1** dejará de hacerlo.

Lo mismo ocurre con las otras compuertas, observa esta con una compuerta NOR, una de sus entradas forma parte del oscilador y la otra hace de Control.



Si lo quieres hacer con compuertas NAND, es igual que el anterior, solo que esta vez un "1" en la entrada de Control habilita al oscilador y un "0" lo inhabilita.

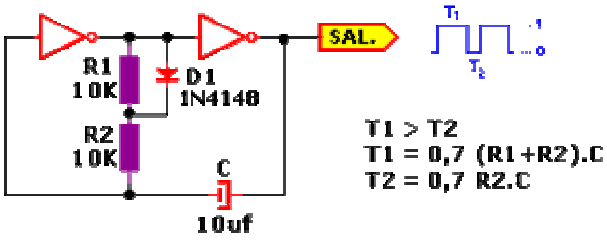


Debes estar cansado ya de tantos osciladores, pero la tentación me mata, el tema es que cierta vez quería controlar la velocidad de un motor de CC y mi única solución era disminuir la tensión lo malo es que también disminuía el torque del motor (fuerza de giro). Hasta que... un día supe que podía controlarla con un circuito astable regulando el ancho de pulso de salida, *como...?*, bueno en la siguiente lección te cuento...

Modulación por ancho de pulso

Introducción - ancho de pulso no simétrico

Nuevamente aquí, a ver si le damos una solución al problema planteado anteriormente, o sea, tratar de que los pulsos de salida no sean simétricos, por ejemplo que el nivel alto en la salida dure más que el nivel bajo, o quizás al revés, bueno veamos el primero.



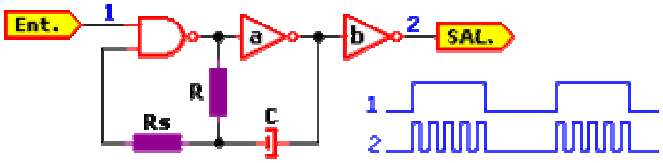
Bien, de entrada ya sabemos que es un circuito astable, solo que esta vez el capacitor se descarga más rápidamente utilizando el diodo como puente y evitando así pasar por **R1**

El efecto obtenido es que **T1** es de mayor duración que **T2**. Puedes ajustar T1 si reemplazas **R1** por un potenciómetro. Los periodos de tiempo para T1 y T2 están dados en la grafica...

Un detalle más... Si inviertes la polaridad del diodo obtendrás la situación inversa, es decir T2 > T1.

Modulación por ancho de pulso Conmutado

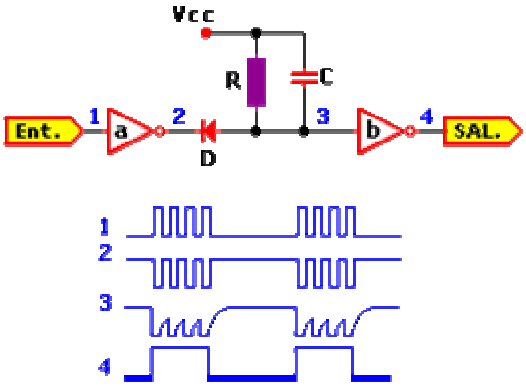
Nada raro... Los mismos circuitos vistos anteriormente pero adaptados para esta tarea. Aquí la cantidad de pulsos de salida depende de la duración del pulso de entrada. Ni para que probar, ya los conocemos y sabemos como funcionan *verdad...?*



Aquel terminal que usábamos antes como terminal de control, ahora está como entrada de señal, y la salida del circuito entregará una cierta cantidad de pulsos mientras dure el pulso de entrada. Si observas la forma de onda en la entrada y la comparas con la salida te darás cuenta de su funcionamiento.

Demodulación de señales

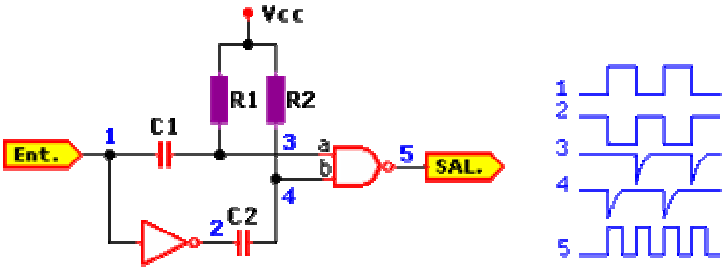
Todo lo opuesto al anterior, es decir tomamos una señal modulada y la demodulamos :o))



Esta vez el tren de pulsos ingresa por el Inversor **a**, en el primer pulso positivo, la salida de **a** se pone a **0** y se carga el capacitor **C** a través del diodo **D**. Cuando la entrada de **a** se invierte el diodo queda bloqueado y **C** se descarga a través de **R**. Ahora bien, durante toda la transmisión de pulsos la salida de **b** permanece a nivel **1** ya que el tiempo de descarga del capacitor es mucho mayor que el tiempo de duración de cada pulso que ingresa por la entrada del inversor **a**

Doblador de frecuencia

Otra aplicación que se pueden dar a las compuertas lógicas es duplicar la frecuencia de una señal, como en este circuito.



Observa la forma de onda obtenidas en los puntos marcados en azul

Analicemos su funcionamiento; El flanco de descenso de la señal de entrada es diferenciada por R1 y C1, y es aplicada a la entrada "a" de la compuerta NAND, esto produce un pulso a la salida de esta compuerta según su tabla de verdad "basta que una de las entradas este a nivel lógico bajo para que la salida vaya a nivel lógico alto"

El flanco de subida del pulso de entrada, luego de ser invertido, es diferenciado y aplicado a la entrada "b" de la compuerta NAND, de modo que para un tren de pulsos de entrada de frecuencia f, hay un tren de pulsos de salida de frecuencia 2f.

Basta de circuitos astables, veamos como hacer un monoestable...

Circuitos Monoestables

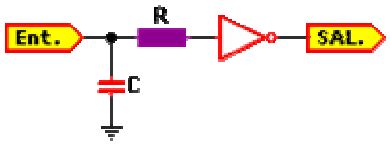
Habrá que recordar de que se trata esto no...?

De acuerdo..., son aquellos que tienen un único nivel de salida estable. Para aclarar un poco las ideas... La mayoría de los edificios disponen de un pulsador que enciende momentáneamente las luces de los pasillos, transcurrido un cierto tiempo éstas se apagan. Conclusión; sólo disponen de un estado estable (apagado) y un estado metaestable (encendido). *se entendió...?*.

Bien, veamos el primero:

Monoestables sencillo

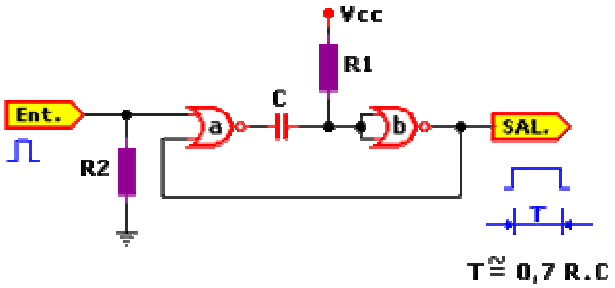
Primero lo básico, un monoestable sencillo con un inversor...



Considera inicialmente la entrada del inversor en nivel bajo a través de **R** y **C**, entonces su salida estará a nivel alto, ahora bien, un **1** lógico de poca duración en la entrada, hace que se cargue el capacitor y conmute el inversor entregando un **0** lógico en su salida, y este permanecerá en ese estado hasta que la descarga del capacitor alcance el umbral de histéresis de la compuerta y entonces conmutará y regresará a su estado inicial...

Monoestables con dos compuertas NOR

Fíjate que la compuerta **b** la puedes cambiar por un inversor...Tratemos ahora de interpretar su funcionamiento



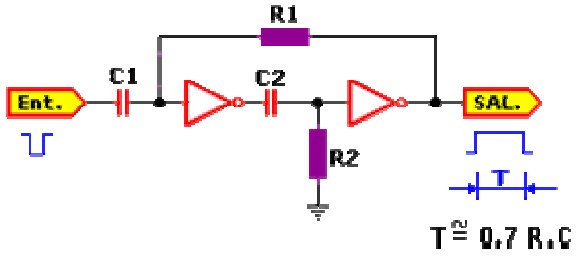
Suponte que no existe señal en la entrada, entonces la compuerta **b** tiene su entrada a nivel "1" por intermedio de **R1**, y su salida a nivel "0", la cual alimenta una de las entradas de **a**. al estar ambas entradas de "a" a nivel "0" la salida de "a" estará a nivel "1". Como el capacitor **C** tiene sus dos extremos al mismo nivel no adquiere carga alguna.

Si entregas un impulso positivo a la entrada de **a**, su salida pasa inmediatamente a nivel "0" y **C** comienza a cargarse a través de **R1**, la entrada de **b** se hace **0** y su salida **1**, como ésta realimenta la compuerta **a** la deja *enganchada* con su salida a **0**.

Cuando la carga del capacitor alcanza el umbral de conmutación de "b" su salida pasa a **0** y la de **a** pasa a **1**, esto hace que el capacitor se descargue a través de R1 y la línea de alimentación, dejando al circuito listo para un nuevo disparo.

Monoestables con dos inversores

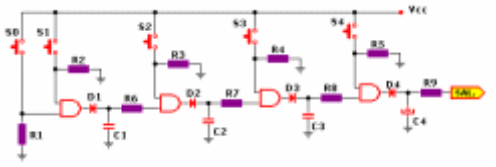
La diferencia aquí, esta en que el gatillado se realiza durante la excursión negativa del pulso de entrada.



Como verás, estos circuitos disponen de algún método de realimentación y un capacitor que es quien retiene momentáneamente una determinada señal lógica en la entrada de alguna de las compuertas implicadas en el circuito...

Cerradura con teclado electrónico

Lo mejor que se me pudo ocurrir para la aplicación de un monoestable fue una cerradura electrónica sencilla, aquí la puedes ver...



La secuencia para activar la salida es el orden en que están numerados los pulsadores, eso sí, nota que debes pulsar S0 y sin liberarlo activar S1, luego de eso puedes continuar con la secuencia correspondiente.

Los botones libres del teclado deberían ir unidos a un sistema de alarma o hacer que se desactive momentáneamente todo el sistema antes de ingresar un nuevo código, en fin tienes muchas opciones. En la salida deberás conectar un relé, un optoacoplador o algo por el estilo para accionar la cerradura electrónica.

En realidad la intención era darle una utilidad a los circuitos monoestables, y esta me pareció ideal

Los componentes utilizados son los siguientes:

D1 a D4 = 1N4148
R1 a R6 = 1k
R7 a R9 = 2k2
C1 a C3 = 1nf
C4 = 1000 uf / 16V
IC1 = CD4081
S1 a S5 = Teclado

Circuitos Biestables

Circuitos Biestables - Parte I

Comencemos... Los circuitos biestables son muy conocidos y empleados como elementos de memoria, ya que son capaces de almacenar un bit de información. Aclarando un poco más, añado que son circuitos que permanecen en un estado hasta que lo cambiamos voluntariamente. En general, son conocidos como Flip-Flop y poseen dos estados estables, uno a nivel alto (1 lógico) y otro a nivel bajo (cero lógico), *Se entendió...?, aplausos para mi... gracias, muchas gracias...!!!*

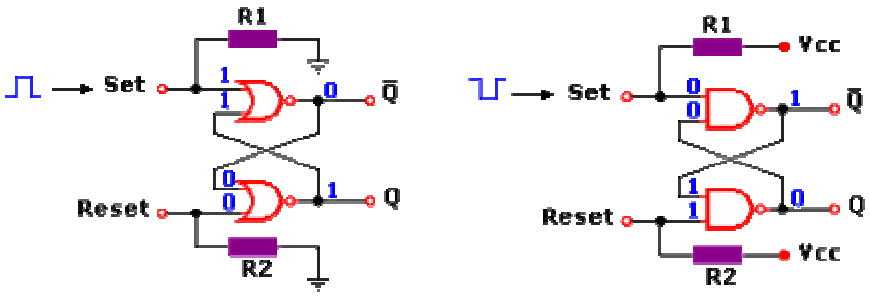
Perdón, me estaba olvidando de un pequeño detalle, es posible que al presionar el pulsador se produzcan rebotes eléctricos, es como haberlo presionado varias veces, y sí... los resultados serán totalmente inesperados, así que lo de los cablecitos para probar estos circuitos no nos servirán de mucho, es conveniente utilizar un pulso de reloj para realizar estas pruebas, *ya sabes...!!! un circuito astable, de los que hicimos en lecciones anteriores*, de ahora en más lo llamaremos **pulso de reloj** o **Clock** o **CK**.

Por lo general un Flip-Flop dispone de dos señales de salida, una con el mismo valor de la entrada y otra con la negación del mismo o sea su complemento.

Primero lo básico, como siempre, y luego lo enredamos un poco más. De momento lo dejamos así, poco a poco veremos como esto que parece complejo es muy simple.

FLIP FLOP BÁSICO RS

Se puede construir uno fácilmente utilizando dos compuertas NAND o NOR conectadas de tal forma de realimentar la entrada de una con la salida de la otra, quedando libre una entrada de cada compuerta, las cuales serán utilizadas para control **Set** y **Reset**...



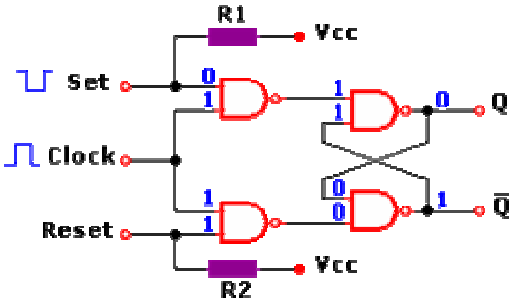
Las resistencias R1 y R2 utilizadas en ambos casos son de 10k y las puse solamente para evitar estados indeterminados, observa el circuito con compuertas NOR... Un nivel alto aplicado en **Set**, hace que la salida negada **~Q** sea **0** debido a la tabla de verdad de la compuerta NOR, al realimentar la entrada de la segunda compuerta y estando la otra a masa, la salida normal **Q** será **1**. Ahora bien, esta señal realimenta la primer compuerta, por lo tanto no importan los rebotes, y el FF se mantendrá en este estado hasta que le des un pulso positivo a la entrada **Reset**

Conclusión: El biestable posee dos entradas **Set** y **Reset** que trabajan con un mismo nivel de señal, provee dos salidas, una salida normal **Q** que refleja la señal de entrada Set y otra **~Q** que es el complemento de la anterior.

Si comparas los dos flip-flop representados en el gráfico, verás que sólo difieren en los niveles de señal que se utilizan, debido a la tabla de verdad que le corresponde a cada tipo de compuerta.

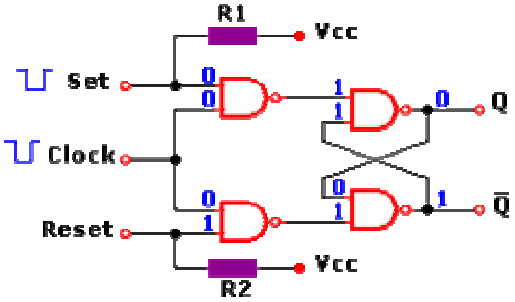
FLIP FLOP RS - Controlado por un pulso de reloj:

En este caso voy a utilizar el ejemplo de las compuertas NAND, pero le agregaremos dos compuertas mas, y uniremos la entrada de cada una a una señal de Reloj...



Lo dicho mas arriba, necesitamos un generador de pulsos (Astable) para conectarlo en la entrada Clock, una vez lo tenemos pasamos a interpretar el circuito...

Si pones un **0** en **Set** y la entrada **Clock** está a **1** ocurrirá todo lo que se describe en el esquema anterior, veamos que ocurre cuando **Clock** pasa a **0**...



Sorpresaaaaaaaaa...!!!, el FF se mantiene sin cambios en **Q** y **~Q**. Fíjate que ahora no importa el estado de **Set** y **Reset**, esto se debe a su tabla de verdad (basta que una de sus entradas sea **0** para que su salida sea **1**) por lo tanto **Set** y **Reset** quedan inhabilitadas.

Es decir que se leerán los niveles de **Set** y **Reset** sólo cuando la entrada **Clock** sea **1**.

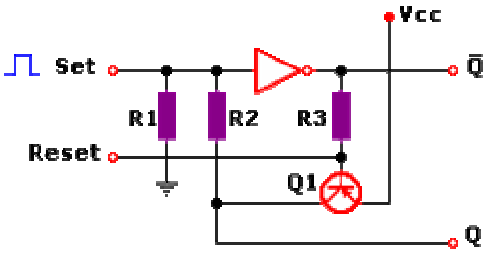
NOTA 1: El primer circuito que vimos (Flip-Flop simple) es llamado **Flip-Flop Asíncrono** ya que puede cambiar el estados de sus salidas en cualquier momento, y sólo depende de las entradas Set y Reset.

NOTA 2: El segundo circuito es controlado por una entrada Clock y es llamado **Flip-Flop Síncrono** ya que el cambio de estado de sus salidas esta sincronizado por un pulso de reloj que realiza la lectura de las entradas en un determinado instante.

Antes de continuar quiero mostrarte algo muy interesante, no es la única forma de obtener un Flip-Flop, observa esto...

FLIP FLOP - Con un inversor

La ventaja aquí es la cantidad de compuertas utilizadas, esta bueno, no te parece...?



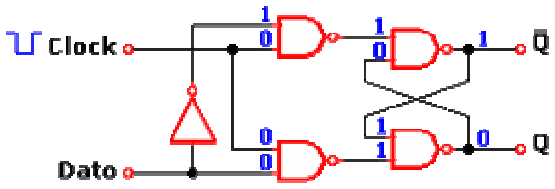
Bueno, lo dejo para que lo analices...

Circuitos Biestables - Parte II

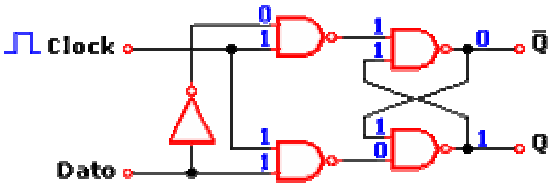
El flip-flop presentado anteriormente conocido como **flip-flop RS** suele presentar un estado indeterminado cuando sus dos entradas R y S se encuentran en estado alto así que veamos como se puede solucionar este inconveniente...

FLIP FLOP D:

En este circuito no existe la posibilidad de que las dos entradas estén a nivel alto, ya que posee un inversor entre una y otra, de tal modo que R = ~S, observa el siguiente gráfico, aquí se supone la entrada Dato a nivel 0...

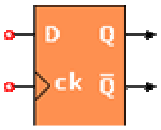


Veamos que ocurre cuando la entrada Dato, pasa a 1 y CK cambia de estado pasando también a 1, según como se van transmitiendo los datos por las compuertas resulta Q=1 y ~Q=0.



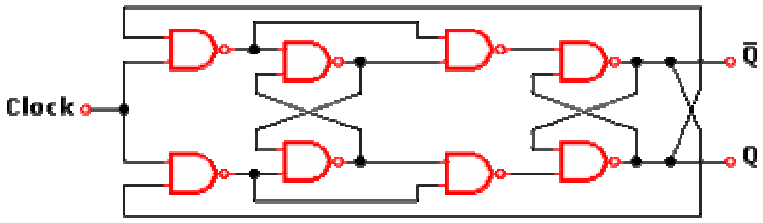
Para que el flip-flop retorne a su estado inicial, la entrada Dato **D** deberá pasar a **0** y sólo se transferirá a la salida si Ck es **1**. Nuevamente se repite el caso que para leer el datos debe ser ck=1.

En forma general se representa el filp-flop D con el siguiente símbolo



FLIP FLOP Master-Slave:

Se trata de un arreglo de dos FF independientes. El primero actúa como Master y el otro como Slave. Con la diferencia de que en este caso las entradas Set y Reset son realimentadas por las salidas Q y ~Q respectivamente, quedando libre únicamente la entrada CK.



Ya se, será complicado de analizar, pero lo haremos fácil, veamos...

Considerando CK=0, será la salida Q=0 y ~Q=1, al momento del cambio de nivel de CK (CK=1), sólo cambiaran las salidas del primer flip-flop (Master) sin afectar las salidas Q y ~Q.

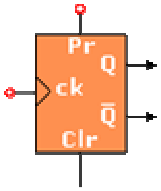
Ahora bien, cuando CK regrese a su estado inicial (CK=0) el Slave conmutará las salidas Q y ~Q quedando Q=1 y ~Q=0. Al cambiar de estado CK (CK=1) las salidas no serán afectadas. Esto se puede resumir en una pequeña tabla de verdad, como ésta...

CK	Q	~Q
0	0	1
1	0	1
0	1	0
1	1	0
⊓	⊓	⌋

Bueno, le agregué una fila más, por si preguntas ;-)

A este tipo de Flip-flop, se le a dado la posibilidad de preestablecer el estado de sus salidas, adicionándole dos entradas más, Preset (**Pr**) y Clear (**Clr**), que vendrían a ser algo así como Set y Reset respectivamente, pero claro, hay que advertir que se debe evitar la situación Pr=Clr=0

También tiene una forma de representación simbólica...

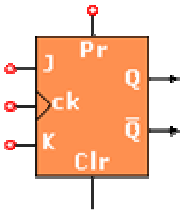


Y aún queda más, el flip-flop JK, así que vamos por él...

Circuitos Biestables - Parte III

FLIP FLOP JK:

Un flip-flop JK es muy similar al visto anteriormente, pero mucho más complejo que éste, y existen Circuitos integrados que ya lo traen incorporado así que por cuestiones de sencillez y para no complicarte demasiado utilizaré su representación simbólica.



Lo vamos a analizar de forma sencilla haciendo uso de la tabla de verdad que corresponde al funcionamiento del flip-flop...

Clr	Pr	J	K	CK	Q	~Q
0	1	X	X	X	0	1
1	0	X	X	X	1	0
1	1	X	X	0	Sc	Sc
1	1	X	X	1	Sc	Sc
1	1	X	X	↑	Sc	Sc
1	1	0	0	X	Sc	Sc
1	1	0	1	↓	0	1
1	1	1	0	↓	1	0
1	1	1	1	↑	Cs	Cs

Comencemos:

Las dos primeras líneas indican que las entradas Clr y Pr establecen el estado de las salidas Q y ~Q sin importar el estado en que se encontraban anteriormente, ni el estado de las otras entradas (J, K y CK).

En la tercera y cuarta línea se han establecido las entradas Clr y Pr a nivel 1 y las salidas Q y ~Q permanecen en cualquiera de los dos estados mencionados anteriormente, según el que se haya establecido...!!!, ahora bien si se mantiene CK=0 las salidas Q y ~Q permanecen sin cambio (Sc), lo mismo ocurre si se mantiene CK=1, y continúa así en los cambios ascendentes de CK, y como podrás notar en la siguiente línea, si estableces J=K=0 queda sin importancia la entrada CK y sin cambio las salidas.

En la séptima y octava línea se transfieren los datos de las entradas J y K a las salidas Q y ~Q respectivamente, pero esto sólo ocurrirá en la transición ascendente de CK.

Finalmente con Clr=Pr=J=K=1 el flip-flop Cambiará Siempre (Cs) cada vez que se presente una transición descendente de CK.

Y hasta aquí..., la idea fue mostrarte las ventajas y desventajas de cada uno de estos circuitos, te recuerdo que no necesitas armar uno de estos embrollos de compuertas, ya que existen integrados que las contienen, como el CD4027 que es un doble flip-flop JK maestro-esclavo o el CD4013 que es un doble flip-flop tipo D, al cual le voy a dedicar una página especial, por sus variadas aplicaciones, muy utilizado en robótica...!!!

Operadores Lógicos en Circuitos Integrados

Es más, estoy pensando que sería bueno describir las funciones de algunos integrados, los más utilizados según el tipo de aplicación y diseño que se requiera, ya veré si me doy tiempo para ello, pero creo que me merezco un descanso, así que..., ahí nos vemos.

Circuitos Integrados C-MOS y TTL – Datos técnicos

Es muy importante disponer de la hoja de datos de los integrados.

Selecciona la compuerta en la tabla correspondiente al tipo de IC (C-MOS ó TTL) del cual deseas ver los datos.

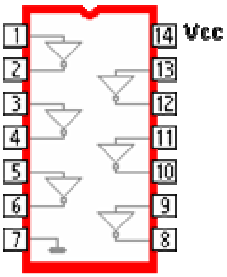
No incluí todos los datos pero creo que estan los necesarios para que puedas trabajar (*al menos por ahora*).

Recuerda los niveles de tensión, yo recomendaría, que no excedas en un integrado C-MOS los 12V, y en un TTL los 5V. Ahora, si dispones de la hoja de datos (del fabricante) del integrado. ya sabras que hacer

Una cosa más... Algunos integrados disponen de salida Buffer, esto mejoran las características de transferencia ya que proveen muy alta ganancia. Otros dispositivos son tipo Disparador Schmitt, (*Esto que es...?*) bueno, básicamente estos últimos pueden reconocer señales un poco distorcionadas y llevarlas a estados lógicos definidos, ya sea Cero o Uno.

Puertas lógicas CMOS

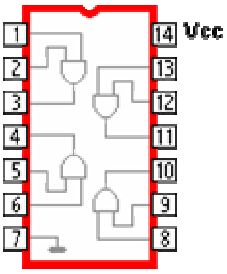
NOT



Séxtuple Circuito Inversor:
4069, CD4069, CD4069UB, MC14069UB, UPC4069UBC

Disparadores Schmitt:
CD40106, CD40106B, MC14584, 40014

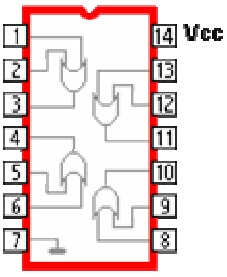
AND



Cuádruple Compuerta AND con Buffer de Salida:

4081B, CD4081, CD4081B, MC14081B

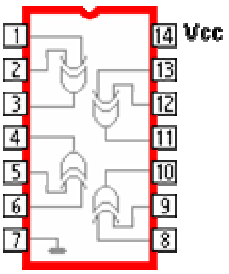
OR



Cuádruple Compuerta OR con Buffer de Salida:

4071B, CD4071, CD4071B, MC14071B

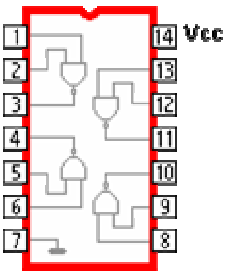
OR-EX



Cuádruple Compuerta OR-EX:

4070B, CD4070, CD4070B, MC14070, 4030B, CD4030, MC14030

NAND

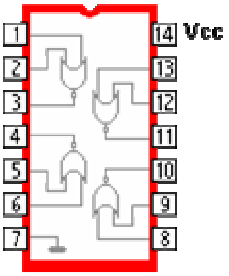


Cuádruple Compuerta NAND: 4011, CD4011, MC14011

Con Buffer de Salida: 4011B, CD4011BC, CD4011BM, MC14011B

Disparadores Schmitt: 4093B, CD4093B, MC14093B

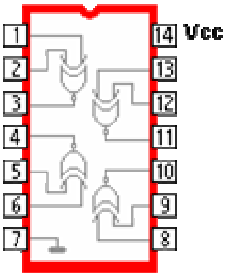
NOR



Cuádruple Compuerta NOR: CD4001, MC14001

Con Buffer de Salida: 4001B, CD4001BC, CD4001BM, MC14001B

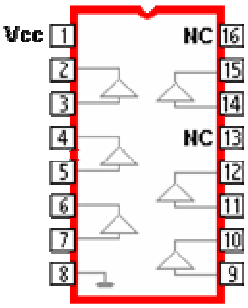
NOR-EX



Cuádruple Compuerta NOR-EX:

CD4077, MC14077

BUFFER

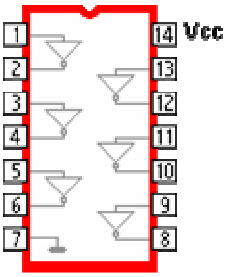


Séxtuple Separador Buffer no inversor:

El representado en el gráfico:
4050B, CD4050BM, CD4050BC, MC14050B, CD4010M, CD4010C.
En los dispositivos 4050 el nivel alto de entrada puede exceder en nivel de alimentación Vcc.
En los 4010 el nivel de salida alcanza a Vcc.
Los pines NC quedan libres (No Connected).
Los pines de los inversores tienen igual posición y recuerde que el símbolo es distinto, 4049B, CD4049, CD4049UB, MC14049UB
CD4009, CD4009M.
Los dispositivos 4049 tienen características similares a los 4050 y los 4009 a los 4010

Puertas lógicas TTL

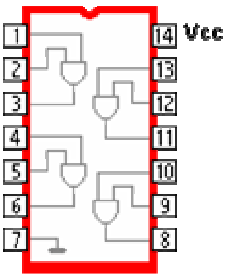
NOT



Séxtuple Inversor sin buffer de salida: SN7404, SN74H04, SN74L04, SN74LS04, SN74S04, SN7405, SN74H05, SN74LS05, SN74S05.

Con Buffer de salida: SN7406, SN7416

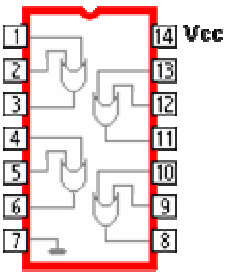
AND



Cuádruple Compuerta AND sin buffer de salida:

SN7408, SN74L08, SN74LS08, SN7409, SN74L09, SN74LS09

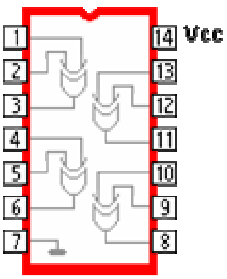
OR



Cuádruple Compuera OR sin buffer de salida:

SN7432, SN74LS32, SN74S32

OR-EX

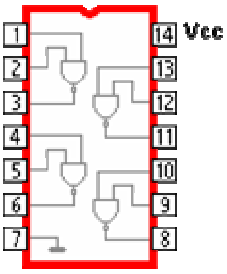


Cuádrupe Compuerta OR-Exclusiva: 7486, 74S86, 74LS86, 74HC86, DL086

Dispositivos con salida de colector abierto: 74136, 74LS136

Vcc=5,5V

NAND



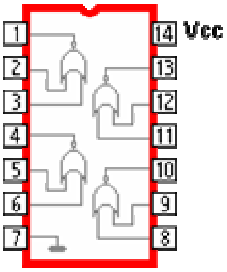
Cuádruple Compuerta NAND:

Sin Buffer de salida: SN7400, SN74H00, SN74L00, SN74LS00 y SN74S00, SNSN74H01, SN7403, SN74L03, SN74LS03, SN74S03

Para interface de alto voltaje(15V): SN7426, SN74LS26

Con Buffer de salida: SN7437, SN74LS37, SN74S37, SN7438, SN74LS38, SN74S38

NOR

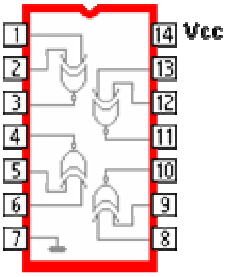


Cuádruple Compuerta NOR:

Sin Salida Buffer: SN7402, SN74L02, SN74LS02, SN74S02

Con Salida Buffer: SN7428, SN74LS28, SN7433, SN74LS33

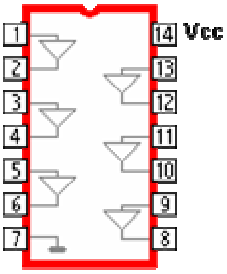
NOR-EX



Cuádruple Compuerta NOR-Exclusiva:

74266, 74HC266, 74LS266, 74AHCT266

Séxtuple Buffers/Drivers:



SN7407, SN7417

Transistores en circuitos de conmutación

Introducción

La verdad es que me hice tal lío, que no sabía si incluir lo de transistores en Electrónica Digital o en la sección de Electrónica Básica, pero como la finalidad era manejar señales lógicas, bueno opte por esta sección.

Se trata de un tutorial de circuitos de conmutación haciendo uso de transistores, incluye un poquitin de cálculo, de todos modos, creo que no les traerá grandes problemas.

Una cosa más, La información disponible en el tutorial de transistores fue extraída de un libro de electrónica digital, publicado por Saber electrónica hace varios años y que me sirvió de mucho y espero que a Uds. también..

Tansistores en circuitos de conmutación

Muchas veces se presenta la difícil situación de manejar corrientes o tensiones más grandes que las que entrega un circuito digital, y entonces nos disponemos al uso de transistores, el tema es hacer que estos trabajen en modo corte y saturación sin estados intermedios, es decir que cambien su estado de plena conducción a un profundo corte, y eso es lo que veremos en este pequeño tutorial.

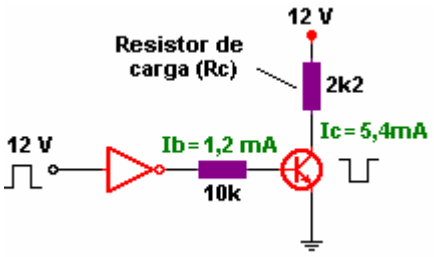
Los transistores a utilizar en estos casos deben tener la suficiente ganancia para que la onda cuadrada, aplicada en su entrada (Base), no sufra ninguna deformación en la salida (Colector o Emisor), o sea que conserve perfecta simetría y sus flancos ascendente y descendente se mantengan bien verticales.

La corriente máxima que puede circular de colector a emisor está limitada por la tensión de polarización de Base y el Resistor o la carga del colector.

Polarización de un transistor NPN como Emisor Común

En este caso el emisor está conectado a masa, se dice que este terminal es común a la señal de base y de colector. El utilizado en este caso un **BC547** y estos son algunos de sus datos:

- Tensión Base-Colector (V_{CBO}) = 50 V
- Corriente de Colector (Ic) = 100mA = 0,1A



Cuando la base de Q1 se polariza positivamente, éste conduce la máxima corriente, que le permite Rc.

Rc es la resistencia de carga, que bien podría ser un LED, un relé, etc.

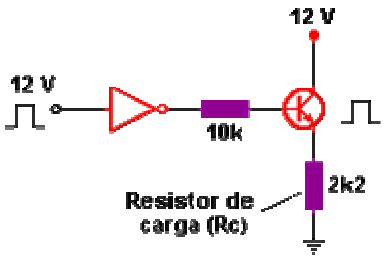
Ic = E/R = 12V / 2200 = 0,0054 = **5,4 mA**

Ib = E/R = 12V / 10000 = 0,0012 = **1,2 mA**

Es decir la corriente total Colector-Emisor es 6,6mA.

Conexión como seguidor emisivo:

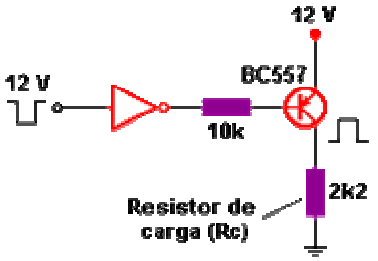
En esta situación se toma la señal de salida desde el Emisor donde se encuentra la Resistencia de carga, observa que este esquema comparado al anterior tiene la misma fase de salida que la de entrada.



También hay casos en que necesitas que el transistor esté conduciendo permanentemente (estado de saturación) y que pase al corte ante la presencia de un pulso eléctrico, esto sería lo inverso de lo visto anteriormente, para lograr esto, los circuitos anteriores quedan como están y sólo se reemplazan los transistores por los complementarios, o sea donde hay un NPN se conecta un PNP.

Cuando la señal es negativa

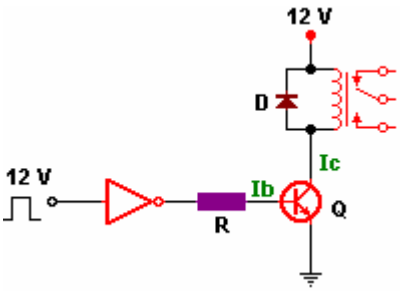
En ocasiones se da el caso en que las señales lógicas recibidas son negativas o de nivel bajo, para entonces se puede utilizar un transistor PNP, por ejemplo: el BC557, que es complementario del BC547, para conseguir los mismos resultados. En la siguiente figura se representa esta condición, es decir, un acoplamiento con transistor PNP.



Análisis para la conexión de un RELE

El diodo en paralelo con la bobina del relé cumple la función de absorber las tensiones que se generan en todos los circuitos inductivos. Es muy importante para la vida del transistor.

Si la bobina del relé tiene 50 ohm de resistencia y funciona a 12 V, puedes calcular el consumo de corriente que tiene el relé, para así saber que transistor utilizar:



Ic = E/R = 12V / 50 = 0,24 = **240 mA**

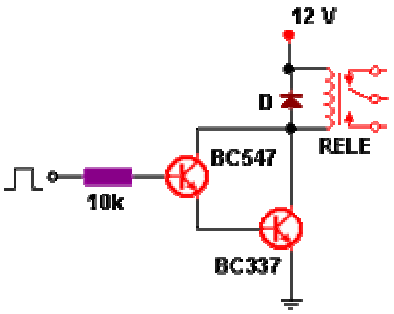
Con este resultado no se puede utilizar el BC547, cuya corriente máxima es de 100mA, pero si lo puede hacer un BC337, es conveniente no superar el 50% de la corriente que entregan los transistores.

Ahora bien, si la señal que se aplique a la base del transistor tiene la suficiente amplitud (tensión) y suficiente intensidad (amper), no habrá dificultad y la corriente de base también será suficiente para saturar el transistor, que conmutará en forma efectiva el relé.

Montajes Darlington

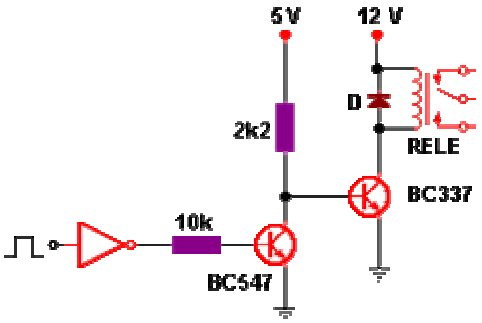
En esta conexión se utiliza un BC337 (NPN) el cual si soporta los 240mA que se necesitaba anteriormente, pero además un transistor de baja potencia como el BC547 (NPN).

En este tipo de montajes, hay que lograr previamente una ganancia en corriente y esta corriente aplicarla a la base del BC337, esta es la finalidad del montaje en Darlington.



En este circuito el Transistor BC337 es el que recibe la carga del relé y el BC547 solamente soporta la corriente de base del BC337, además la ganancia se multiplica sin cargar la salida del componente que entrega la señal, ya que ahora la corriente que drena el 547 es tomada de la misma fuente y aplicada a la base del 337. De este modo la resistencia de base del 547 puede ser elevada ya que necesitamos una corriente mucho menor en la misma.

En el siguiente gráfico se describe como lograr la conmutación de un relé con un **transistor de salida** NPN, incluso utilizando tensiones diferentes.



En esta situación como vemos es necesario agregar un transistor de baja potencia, ya que la corriente que debe manejar es la de base.

Con la entrada en "1": El BC547 conduce y envía a masa la base del BC337 de este modo se mantiene el corte.

Con la entrada en "0": El 547 pasa al corte y su colector queda "abierto", ahora sí se polariza la base del 337 y conmutando el relé.

Otro caso de conmutación con diferentes tensiones.

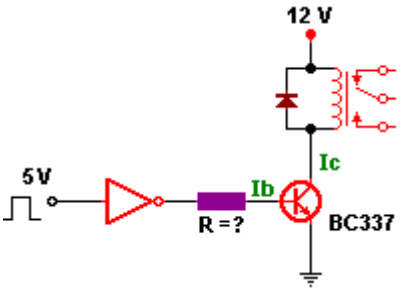
Suponiendo que el consumo de un relé sea 200mA.

Para los cálculos de polarización siempre se debe tomar el menor Beta-B-(hfe) que indiquen los manuales de los transistores, o sea que si dice 100 a 300, tomamos 100. Veamos que corriente de base se necesita de acuerdo a estos datos:

Ib = Ic / Hfe = 200mA / 100 = **2mA**

Donde:

- **Ib** = Intensidad de Base (en mA)
- **Ic** = Intensidad de Colector
- **Hfe** = Ganancia



Ahora veamos que valor de resistencia de base es necesario para lograr 2mA con una fuente de 5V, que es la salida que entrega el separador del ejemplo

R = E / I = 5V / 0,002A = **2500 ohm** (un valor normalizado es 2k2)

Hay circuitos más complejos todavía, pero creo que puede servir de algo tener un poco de conocimiento de estas formas de conexión. Recuerden, no se olviden del diodo conectado en paralelo con el relé.

Puerto paralelo

Introducción

No podía ser de otra manera, la verdad, ja! breve...!!!, sólo por ahora, pronto verás hasta donde llegaremos...

En fin, así es como nos iniciaremos, con 5 lecciones describiendo algunas de las características de este puerto y sus posibles aplicaciones, es bueno tener algún conocimiento de electrónica digital, por si no lo tienes consulta la página correspondiente, o busca otras si con ésta no te es suficiente, sabes que Internet dispone de muchísimo material al respecto.

Quiero aclarar algo, cualquier daño ocasionado a tu ordenador es sólo tu responsabilidad, por mi parte los circuitos y programas que se encuentran en esta página fueron probados en mi AMDk5, y todavía vive...!!!

Por esta razón, no me vengas con quejas, de acuerdo...?

Algunos circuitos fueron extraídos de la www.... pero solo los que más me gustaron están aquí, al menos conmigo funcionaron. Los programas que no son de mi autoría llevan el nombre de sus Creadores y no pretendo lucrar con ellos ni mucho menos, lo que sí creo, deberíamos hacer todos es agradecerles el valioso tiempo invertido en esa tarea, y lo que es más, colocarlos en la red a disposición de todos nosotros. Saludos para todos ellos...!!!

Cualquier mensaje apreciación o comentario ya sabes, a mi correo...!!! y si deseas contactar conmigo también.

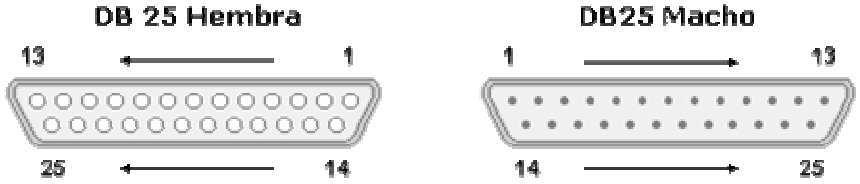
Bueno, ya está, ahora sí podemos comenzar...

Conectores del Puerto Paralelo (DB25).

En verdad se merecía toda una sección aparte a la cual le brindaremos una atención especial e iremos ampliándola cada vez más, ya que sus prestaciones son muchísimas, y es bueno que lo conozcas, te sorprenderás de las cosas que se pueden hacer...

Si estas leyendo este tutorial es por que tienes una PC en estos momentos y seguramente una impresora o un Scanner conectados a ella. Bien, la ficha en la cual están conectados estos dispositivos es conocida como puerto paralelo. Si la desconectas veras que tiene 25 contactos (pines de ahora en más), su nombre es Conector DB25.

Aquí tienes una imagen de los conectores



Cada pin de los conectores tiene un número asignado por lo cual es muy importante que al armar tus propios cables los identifiques correctamente o sufrirás unos terribles dolores de cabeza...!!!

Registros del Puerto Paralelo

Este puerto dispone de tres registros de 8 bit cada uno (un byte).

Que no sabes lo que es un bit o un byte...?, de acuerdo, ingresa **aquí** te sacas las dudas y regresas...

Sigamos con lo nuestro... cada uno de estos registros se denominan puertos o PORT., y cada uno de sus bits, representa un pin determinado del puerto. Los pin's que van del 18 al 25 (ambos inclusive): Son para masa, y sirven para conectar las descargas de los circuitos.

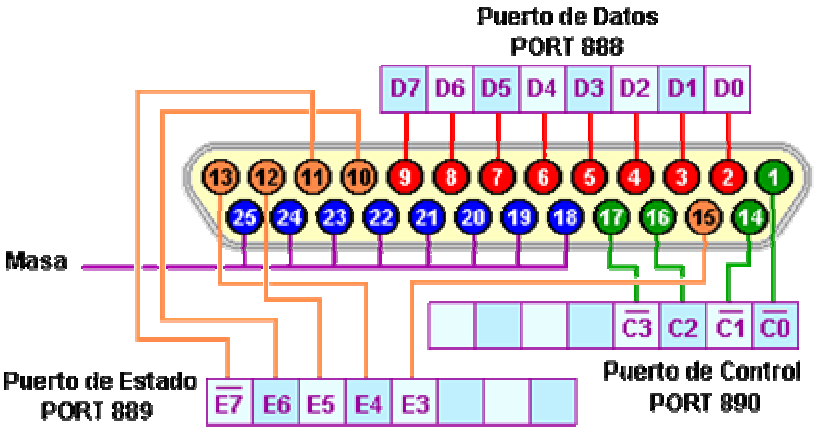
Veamos ahora los tres registros

Puerto de datos (Pin 2 al 9): Es el PORT 888 y es de solo escritura, por este registro enviaremos los datos al exterior de la pc, cuidado...!!!, no envíes señales eléctricas al ordenador por estos pines.

Puerto de estado (Pin 15, 13, 12, 10 y 11): Es el PORT 889 y es de solo lectura, por aquí enviaremos señales eléctricas al ordenador, de este registro solo se utilizan los cinco bits de más peso, que son el bit 7, 6, 5, 4 y 3 teniendo en cuenta que el bit 7 funciona en modo invertido.

Puerto de control (Pin 1, 14, 16 y 17): Es el correspondiente al PORT 890, y es de lectura/escritura, es decir, podremos enviar o recibir señales eléctricas, según nuestras necesidades. De los 8 bits de este registro solo se utilizan los cuatro de menor peso o sea el 0, 1, 2 y 3, con un pequeño detalle, los bits 0, 1, y 3 están invertidos.

En esta imagen puedes ver los tres registros, sus bits y los pines asignados a cada uno de ellos. La imagen corresponde a un conector DB-25 (Hembra).


Puerto de Datos
PORT 888
D7 D6 D5 D4 D3 D2 D1 D0
13 12 11 10 9 8 7 6 5 4 3 2 1
25 24 23 22 21 20 19 18 17 16 15 14
Masa
C3 C2 C1 C0
Puerto de Control
PORT 890
Puerto de Estado
PORT 889
E7 E6 E5 E4 E3

Algo más... La tensión de trabajo del puerto es de 5 voltios, por lo que necesitamos una fuente estabilizada o regulada de tensión, esto es importante tenerlo en cuenta, ya que estaremos enviando señales al puerto. Por otro lado, si bien puedes utilizar la PC para enviar señales al exterior sin necesitad de una fuente externa, es recomendable utilizarla y así no exiges demasiado al puerto y te evitas problemas.

Ahora bien, si activas un bit de salida por el puerto, este permanecerá así hasta que lo cambies, es decir que estarás enviando 5V de forma continua hasta que lo pongas a 0.

Pero bueno.. eso es programación, y lo veremos a su debido momento, por lo pronto deberemos armarnos de unos circuitos para verificar el funcionamiento del puerto. Aclarado esto pasemos a su construcción...

Enviando señales por el Registro de Datos

Para comenzar construiremos un circuito que nos permita enviar señales por el puerto paralelo y lo haremos utilizando diodos LEDs, como el consumo de los LEDs es superior al que nos envía el puerto utilizaremos un buffer como el 74HC244 o el 74HCT245, prefiero este último ya que la construcción de la placa será mas sencilla, aquí tienes los datos del integrado

Non-inverting 3-state outputs

74HCT245

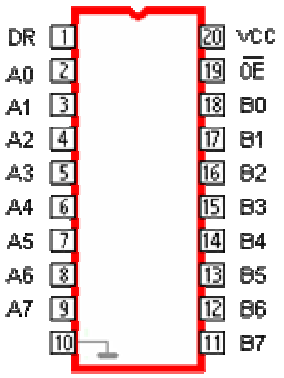


Tabla de Verdad

INPUTS		INPUTS/OUTPUTS	
$\overline{OE}$	DIR	A_n	B_n
L	L	A = B	inputs
L	H	inputs	B = A
H	X	Z	Z

Nota:
X = Sin Importancia
Z = Alta impedancia

Este integrado tiene la ventaja de ser bidireccional, es decir todos los pin's A pueden ser entradas y los B salidas si DIR (pin1 DR) se encuentra a nivel bajo (L), el modo invertido se obtiene con DR a nivel alto. Si el terminal OE (pin 19) se encuentra a nivel alto (H) el integrado pone los pin's A y B con alta impedancia, es decir ni entra ni sale señal alguna, en ese caso el estado de DR no tiene importancia. En la siguiente tabla tienes los nombres de los pines y sus funciones correspondientes.

No. PIN	Nombre	Función
1	DIR	Control de dirección
2, 3, 4, 5, 6, 7, 8, 9	A0-A7	Entrada/Salida de datos
10	GND	Fuente (0V)
18, 17, 16, 15, 14, 13, 12, 11	B0-B7	Entrada/Salida de datos
19	$\overline{OE}$	Habilitación (Activo (L))
20	Vcc	Fuente (+5V)

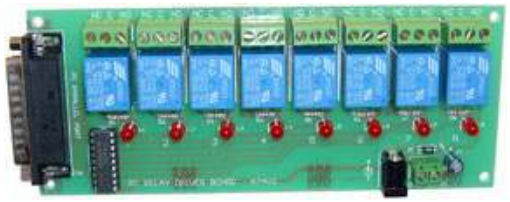
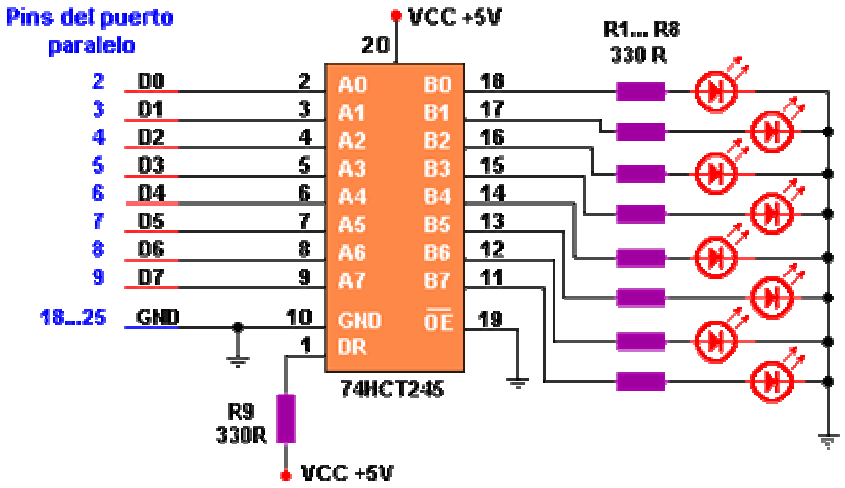


Foto. placa de relés para el puerto paralelo

En el siguiente esquema no se representaron todos los pines del puerto, sino los correspondientes al puerto de datos y los de masa, que son los que utilizaremos en esta primera práctica...



Lo que queda ahora es ver como funciona, y lo haremos sencillito, para ello recurrimos a nuestros superiores en programación *ya que de eso yo no se nada... :o))*.

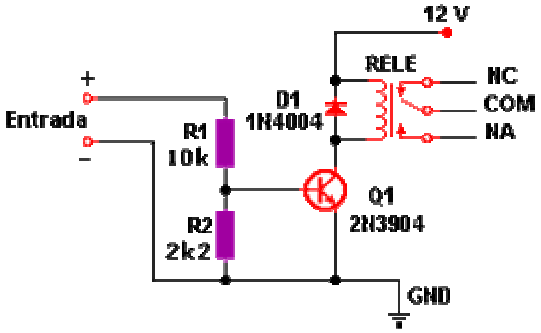
Es broma...!!! Para no meterte en líos con las sentencias de programación las cuales puedes programarlas tu mismo en Basic, en Pascal, en C, en Delphi y todos los demás lenguajes que quieras, yo hice uno en **C** y este es el código fuente...

```
/*Programa que envía señales al puerto de datos ingresando un número
decimal*/

#include<dos.h>
#include<stdio.h>
#include<conio.h>

int a;                  /*variable, valor que enviare al puerto*/
void main()
{
a=0;                    /*asignando 0 a la variable a*/
outp(888,a);            /*apago todos los LED's*/
clrscr();               /*limpio la pantalla*/

printf("Ingrese el número en decimal para enviar al puerto.\n");
printf("El número máximo permitido es 255:\n");

scanf("%d",&a);         /*tomo el valor y se lo paso a "a"*/
outp(888,a);            /*lo envío al puerto de datos*/

printf("tachannnn... Dato enviado...!!!\n\n");
printf("Se acabó, presione una tecla para salir");

getch();                /*...y hasta luego*/
outp(888,0);            /*apago todos los LED's*/
}
```

Creo que no es necesario comentarlo, el ejecutable está en el archivo que podeis descargar en el enlace que tienen al final de este manual, en el apartado descargas y para los amantes de VBasic, un paquete...!!! con código fuente, comentarios y el ejecutable...

Si tienes prisa: http://r-luis.xbot.es/puerto/archivos/pdatos.zip

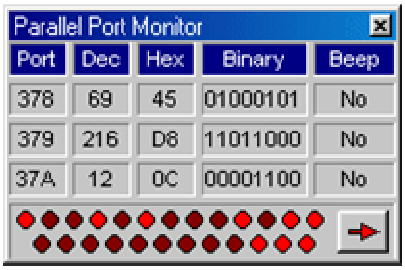
Deseas encender algún aparato de casa...? No hay problema, sólo le agregas una etapa de potencia a cada salida, algo como ésto...



Antes de que me olvide... Me encontré con un pequeño programa que te chequea el estado del puerto, lo puedes bajar de la página de su autor, búscalo en la siguiente dirección web: http://www.geekhideout.com/parmon.shtml, en concreto, el programa es Parallel Port Monitor, también puedes bajarlo desde los enlaces al final del libro.

Monitor de puerto paralelo

El monitor de puerto paralelo es una utilidad para ver y manipular el estado de un puerto paralelo en un equipo con Windows 95/98/ME/NT/2000



Es posible encontrar otros programas que cumplan todos los requisitos que necesitamos para chequear el puerto paraleo, este es muy sencillo y la verdad, para qué complicarnos si este ya nos funciona bien.

Y con esto es suficiente, ya no necesitaras armar los circuitos para verificar tus programas, aunque si deseas utilizar el puerto, no hay de otra, tendrás que hacerlos.

Enviando señales por el puerto de control

Este puerto es de lectura-escritura. En este caso lo utilizaremos para enviar señales desde la PC hacia el exterior, y el circuito será similar al empleado en el puerto de datos, de ese modo es posible tener 12 salidas, lo que si se debe tener en cuenta, es que el primero, el segundo y el cuarto bit de este registro están invertidos, pero esta dificultad es superada con la codificación del programa que controla este puerto.

Si deseas ver como funcionan las salidas, monta el circuito anterior y ya podrás observar las variaciones en los led's.

El programa de prueba lo tienes aquí.

```
/*Programa que envía señales al puerto de control ingresando un número
decimal*/

#include<dos.h>
#include<stdio.h>
#include<conio.h>

int a;                  /*variable, valor que enviare al puerto*/
void main()
{
a=11;                   /*11 en binario es 1011 los valores 1 corresponden*/
                        /*a los pines invertidos*/
outp(890,a);            /*apago todos los LED's*/
clrscr();               /*limpio la pantalla*/

printf("Ingrese el número en decimal para enviar al puerto.\n");
printf("El número máximo permitido es 15:\n");

scanf("%d",&a);         /*tomo el valor y se lo paso a "a"*/
outp(890,a);            /*lo envío al puerto de estado*/

printf("tachannnn... Dato enviado...!!!\n\n");
printf("Se acabó, presione una tecla para salir");

getch();                /*...y hasta luego*/
outp(890,11);           /*apago todos los LED's*/
}
```

Tendré que hacer algunas aclaraciones...?, no creo verdad...? bueno, sólo ten en cuenta los bits invertidos cuando pruebes este programa, que por cierto aquí lo tienes ya compilado se llama **Pcontrol.exe**.

Y como no podía faltar, para los amantes de Visual Basic también les dejo el código, el exe y sus comentarios. Como siempre…. El enlace al final del manual y si tienes prisa:

http://r-luis.xbot.es/puerto/archivos/pcontrol.zip

Algo que olvidaba es la distribución de los pines de control, para que sueldes en el conector DB25, y son los siguientes.

C0 --> pin 1
C1 --> pin 14
C2 --> pin 16
C3 --> pin 17
Masa --> pin 18 al 25

Bien... Con ésto terminamos el envío de señales, luego ampliaremos más, por ahora hasta aquí, espero logres familiarizarte con este tema ya que tiene más posibilidades de las que vimos hasta ahora, te sorprenderías de las cosas que se pueden hacer...!

Pasemos a otro tema, la recepción de señales...!!!

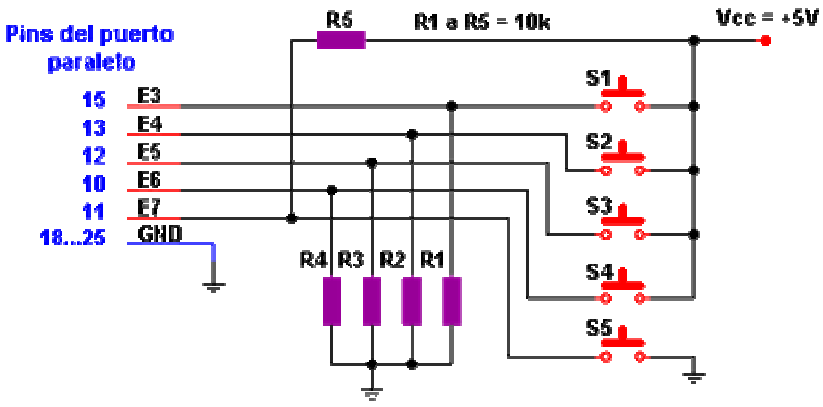
Recibiendo señales por el Puerto Paralelo

Ahora si ya estamos en condiciones de continuar...

Para recibir datos en nuestro ordenador por el puerto paralelo utilizaremos su registro de estado, recuerda que solo dispones de 5 bits (7, 6, 5, 4 y 3) y que el bit 7 funciona en modo invertido.

El circuito que construiremos será uno de los mas sencillos, solo habrá que tener en cuenta que un bit puede tener dos estados posibles, ya sea 0 (0 voltios) o 1 (5 voltios) no podemos dejarlo librado al azar, razón por la cual, si no se envía señal alguna, éste deberá estar unido a masa, de acuerdo...?

El esquema eléctrico es el siguiente;



Observa que para el bit 7 la conexión es invertida si lo pones a 0 el programa lo leerá como un 1, si no presionas el pulsador se leerá como 0 lógico.

Con respecto al programa, para no liar en lenguajes de programación hice uno que nos verifica el estado del puerto, puedes bajarlo si lo deseas, el enlace está al final de este manual y si no puedes esperar porque urge ver el programa:

http://r-luis.xbot.es/puerto/archivos/pestado.zip

Luego podrás hacer uno mucho más interesante que éste, eso espero...!!!

Pues bien, supongo que sabrás que el ordenador manda los datos en forma de bytes. Un byte son 8 bits y un bit es la mínima cantidad de información que un ordenador puede tratar. Es decir, un bit puede tomar sólo dos valores posibles, que son 0 ó 1.

Numeración Binaria y Bits

Hablando en lenguaje eléctrico el 0 significa 0 voltios y el uno significa 5 voltios *(por aquello de la electrónica digital)*. Cualquier valor que mandemos al puerto, el ordenador lo transformará en ceros y unos. Esto se conoce como **sistema binario de numeración**. Como tenemos datos de 8 bits (1 byte), aplicando algo de las matemáticas que aprendimos en EGB, vemos que con sólo 2 dígitos (el cero y el uno) podremos representar 2^8 valores, haciendo números tendremos 256 valores, que se corresponden a la siguiente tabla:

Decimal.	Binario.
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000
9	00001001
etc. etc. etc.	etc. etc. etc.
255	11111111

Bueno, espero que lo hayas comprendido y si no es así, continúa leyendo esta sección porque esto ya es máscomplicado y te servirá para entrar fácilmente en el mundo de la informática.

Numeración binaria, BIT, Byte y Nibbles - En detalle...!!!

.: Bit.

El ordenador se compone de dispositivos electrónicos digitales, por lo tanto éstos solo pueden adoptar únicamente dos estados, que representamos matemáticamente por 0 y 1. Cualquiera de estas unidades de información se denomina BIT, *contracción de «binary digit» en inglés.*

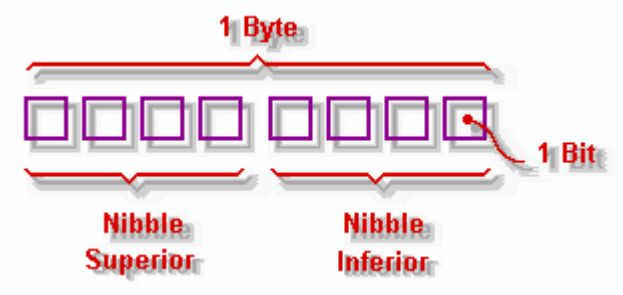
.: Byte.

Cada grupo de 8 bits se conoce como byte u octeto. Es la unidad de almacenamiento en memoria, la cual está constituida por un elevado número de posiciones que almacenan bytes. La cantidad de memoria de que dispone un sistema se mide en Kilobytes (1 Kb = 1024 bytes), en Megabytes (1 Mb = 1024 Kb), Gigabytes (1 Gb = 1024 Mb), Terabytes (1 Tb = 1024 Gb) o Petabytes (1 Pb = 1024 Tb).

Los bits en un byte se numeran de derecha a izquierda y de 0 a 7, correspondiendo con los exponentes de las potencias de 2 que reflejan el valor de cada posición. Un byte nos permite, por tanto, representar 256 estados (de 0 a 255) según la combinación de bits que tomemos.

.: Nibble.

Cada grupo de cuatro bits de un byte constituye un nibble, de forma que los dos nibbles de un byte se llaman nibble superior (el compuesto por los bits 4 a 7) e inferior (el compuesto por los bits 0 a 3).



Un bit es la posición que ocupa un número el cual será "0" o "1" ya que son los únicos valores que admite.

Si dispones de un bit solo tienes 2 posibilidades

1ra. posibilidad	1
2da. posibilidad	0

Si dispones de 2 bits tienes 4 posibles combinaciones.

1ra. combinación	00
2da. combinación	01
3ra. combinación	10
4ta. combinación	11

En fin si dispones de 4 bit tienes 16 posibles combinaciones que son las siguientes:

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

Pues bien, estas son las que corresponden a un Nibble, esto es muy importante ya que cada nibble representa una cifra en el sistema hexadecimal que van desde el 0 al 9 y desde la A a la F

Nibble en binario	Valor Hexadecimal	Valor Decimal
0000	0	0
0001	1	1
0010	2	2
0010	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Existe una forma sencilla de saber cuantas posibles combinaciones puedes obtener con una determinada cantidad de bits, como se trata de un sistema binario de numeración, este se organiza en base 2, entonces 2^n nos da la cantidad de combinaciones que podemos realizar, (**n** en este caso, y solo en este caso lo tomaremos como la cantidad de bit disponibles, recuerda que solo es en este caso, esta...?) veamos un ejemplo;

Al disponer de 1 bit tendremos 2^1 = 2 combinaciones posibles
Al disponer de 2 bits tendremos 2^2 = 4 combinaciones posibles
Al disponer de 3 bits tendremos 2^3 = 8 combinaciones posibles
Al disponer de 4 bits tendremos 2^4 = 16 combinaciones posibles
Al disponer de 5 bits tendremos 2^5 = 32 combinaciones posibles
Al disponer de 6 bits tendremos 2^6 = 64 combinaciones posibles
Al disponer de 7 bits tendremos 2^7 = 128 combinaciones posibles
Al disponer de 8 bits tendremos 2^8 = 256 combinaciones posibles

En este ultimo caso estamos hablando de un byte (son los 8 bits que forman los dos nibbles).

El sistema de numeración hexadecimal agrupa los bits de a cuatro, es por eso que aparecen los nibbles (grupos de 4 bits), observa esta equivalencia de ejemplo; y verifica que sea verdad.

BYTE	**Valor hexadecimal**
0111 0101	75

El puerto Serie

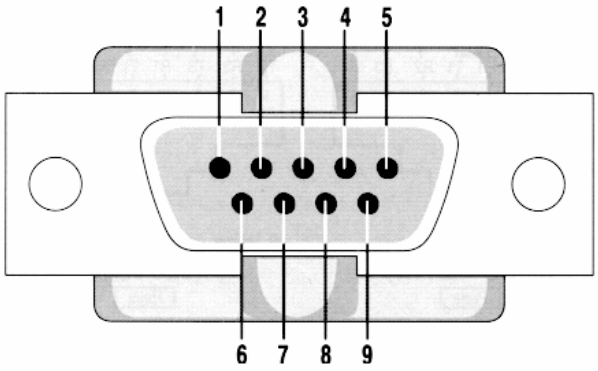
El estándar de comunicaciones RS-232

Como no podía faltar en un manual tan extenso en electrónica y al tener contacto entre el PC y algúnos programadores, es importante un pequeño repaso por el puerto de comunicaciones que usa el PC y algúnas máquinas para conectarse con él.

Un puerto serie o puerto serial es una interfaz de comunicaciones de datos digitales, frecuentemente utilizado por computadoras y periféricos, donde la información es transmitida bit a bit enviando un solo bit a la vez, en contraste con el puerto paralelo que envía varios bits simultáneamente. La comparación entre la transmisión en serie y en paralelo se puede explicar usando una analogía con las carreteras. Una carretera tradicional de un sólo carril por sentido sería como la transmisión en serie y una autovía con varios carriles por sentido sería la transmisión en paralelo, siendo los vehículos los bits que circulan por el cable.

Cómo funciona el puerto serie

El puerto serie RS-232C, presente en todos los ordenadores actuales y algúnas máquinas industriales, es la forma mas comúnmente usada para realizar transmisiones de datos entre ordenadores. El RS-232C es un estándar que constituye la tercera revisión de la antigua norma RS-232, propuesta por la EIA (Asociaci¢n de Industrias Electrónicas), realizándose posteriormente un versión internacional por el CCITT, conocida como V.24. Las diferencias entre ambas son mínimas, por lo que a veces se habla indistintamente de V.24 y de RS-232C (incluso sin el sufijo "C"), refiriéndose siempre al mismo estándar.



Descripcion de Pines

Pin	Signal	E/S	Pin	Signal	E/S
1	DCD – Data Carrier Detect	E	6	DSR - Data Set Ready	E
2	RXD- Received Data	E	7	RTS-Resques to Send	S
3	TXD - Transmited Data	S	8	CTS- Clear to Send	E
4	DTR-Data Terminal Ready	S	9	RI- Ring Indicator	E
5	GND - Signal Ground	⇗			

El RS-232C consiste en un conector tipo DB-25 de 25 pines, aunque desde hace años sólo se utiliza la versión de 9 pines DB-9, mas barato e incluso mas extendido para cierto tipo de periféricos (como el ratón serie del PC). En cualquier caso, los PCs no suelen emplear mas de 9 pines en el conector DB-25. Las señales con las que trabaja este puerto serie son digitales, de +12V (0 lógico) y -12V (1 lógico), para la entrada y salida de datos, y a la inversa en las señales de control. El estado de reposo en la entrada y salida de datos es -12V. Dependiendo de la velocidad de transmisión empleada, es posible tener cables de hasta 15 metros.

Cada pin puede ser de entrada o de salida, teniendo una función específica cada uno de ellos. Las más importantes son:

Conectores serie

Pin	Función
TXD	(Transmitir Datos)
RXD	(Recibir Datos)
DTR	(Terminal de Datos Listo)
DSR	(Equipo de Datos Listo)
RTS	(Solicitud de Envío)
CTS	(Libre para Envío)
DCD	(Detección de Portadora)

Las señales TXD, DTR y RTS son de salida, mientras que RXD, DSR, CTS y DCD son de entrada. La masa de referencia para todas las señales es SG (Tierra de Señal). Finalmente, existen otras señales como RI (Indicador de Llamada), y otras poco comunes que no se explican en este artículo por rebasar el alcance del mismo.

Numero	de Pin	Señal	Descripción	E/S
En DB-25	En DB-9			
1	1	-	Masa chasis	-
2	3	TxD	Transmit Data	S
3	2	RxD	Receive Data	E
4	7	RTS	Request To Send	S
5	8	CTS	Clear To Send	E
6	6	DSR	Data Set Ready	E
7	5	SG	Signal Ground	-
8	1	CD/DCD	(Data) Carrier Detect	E
15	-	TxC(*)	Transmit Clock	S
17	-	RxC(*)	Receive Clock	E
20	4	DTR	Data Terminal Ready	S
22	9	RI	Ring Indicator	E
24	-	RTxC(*)	Transmit/Receive Clock	S

(*) = Normalmente no conectados en el DB-25

El puerto serie en el PC

El ordenador controla el puerto serie mediante un circuito integrado especifico, llamado UART (Transmisor-Receptor-Asíncrono Universal). Normalmente se utilizan los siguientes modelos de este chip: 8250 (bastante antiguo, con fallos, solo llega a 9600 baudios), 16450 (versión corregida del 8250, llega hasta 115.200 baudios) y 16550A (con buffers de E/S). A partir de la gama Pentium, la circuiteria UART de las placa base son todas de alta velocidad, es decir UART 16550A. De hecho, la mayoría de los módems conectables a puerto serie necesitan dicho tipo de UART, incluso algunos juegos para jugar en red a través del puerto serie necesitan de este tipo de puerto serie. Por eso hay veces que un 486 no se comunica con la suficiente velocidad con un PC

Pentium... Los portátiles suelen llevar otros chips: 82510 (con buffer especial, emula al 16450) o el 8251 (no es compatible).

Para controlar al puerto serie, la CPU emplea direcciones de puertos de E/S y líneas de interrupción (IRQ). En el AT-286 se eligieron las direcciones 3F8h (o 0x3f8) e IRQ 4 para el COM1, y 2F8h e IRQ 3 para el COM2. El estándar del PC llega hasta aquí, por lo que al añadir posteriormente otros puertos serie, se eligieron las direcciones 3E8 y 2E8 para COM3-COM4, pero las IRQ no están especificadas. Cada usuario debe elegirlas de acuerdo a las que tenga libres o el uso que vaya a hacer de los puertos serie (por ejemplo, no importa compartir una misma IRQ en dos puertos siempre que no se usen conjuntamente, ya que en caso contrario puede haber problemas). Es por ello que últimamente, con el auge de las comunicaciones, los fabricantes de PCs incluyan un puerto especial PS/2 para el ratón, dejando así libre un puerto serie.

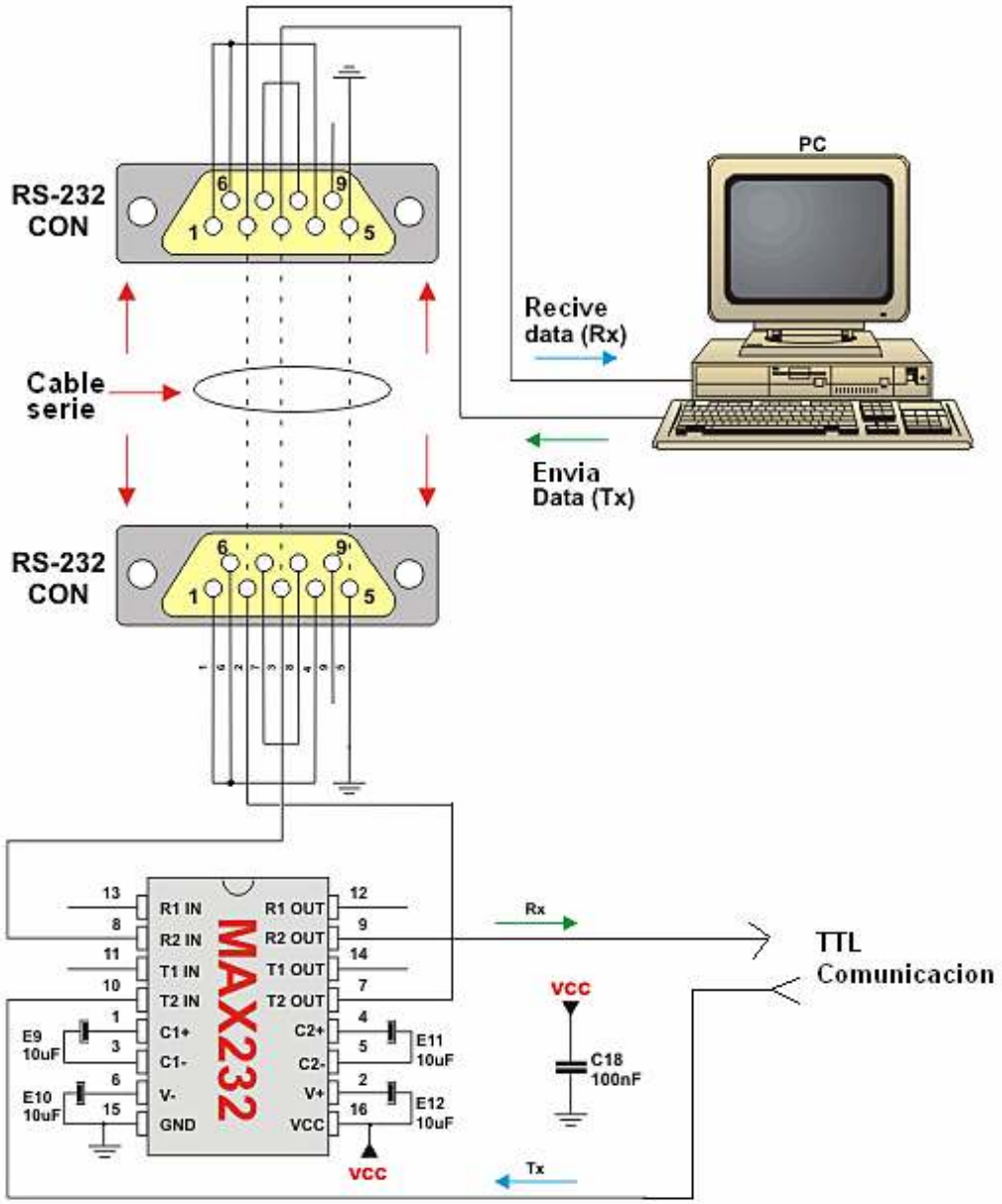
Mediante los puertos de E/S se pueden intercambiar datos, mientras que las IRQ producen una interrupción para indicar a la CPU que ha ocurrido un evento (por ejemplo, que ha llegado un dato, o que ha cambiado el estado de algunas señales de entrada). La CPU debe responder a estas interrupciones lo mas rápido posible, para que de tiempo a recoger el dato antes de que el siguiente lo sobrescriba. Sin embargo, las UART 16550A incluyen unos buffers de tipo FIFO, dos de 16 bytes (para recepción y transmisión), donde se pueden guardar varios datos antes de que la CPU los recoja. Esto también disminuye el numero de interrupciones por segundo generadas por el puerto serie.

El RS-232 puede transmitir los datos en grupos de 5, 6, 7 u 8 bits, a unas velocidades determinadas (normalmente, 9600 bits por segundo o mas). Después de la transmisión de los datos, le sigue un bit opcional de paridad (indica si el numero de bits transmitidos es par o impar, para detectar fallos), y después 1 o 2 bits de Stop. Normalmente, el protocolo utilizado ser 8N1 (que significa, 8 bits de datos, sin paridad y con 1 bit de Stop).

Una vez que ha comenzado la transmisión de un dato, los bits tienen que llegar uno detrás de otro a una velocidad constante y en determinados instantes de tiempo. Por eso se dice que el RS-232 es asíncrono por caracter y sincrono por bit. Los pines que portan los datos son RXD y TXD. Las demás se encargan de otros trabajos: DTR indica que el ordenador esta encendido, DSR que el aparato conectado a dicho puerto esta encendido, RTS que el ordenador puede recibir datos (porque no esta ocupado), CTS que el aparato conectado puede recibir datos, y DCD detecta que existe una comunicación, presencia de datos.

Tanto el aparato a conectar como el ordenador (o el programa terminal) tienen que usar el mismo protocolo serie para comunicarse entre si. Puesto que el estándar RS-232 no permite indicar en que modo se esta trabajando, es el usuario quien tiene que decidirlo y configurar ambas partes. Como ya se ha visto, los parámetros que hay que configurar son: protocolo serie (8N1), velocidad del puerto serie, y protocolo de control de flujo. Este ultimo puede ser por hardware (el que ya hemos visto, el handshaking RTS/CTS) o bien por software (XON/XOFF, el cual no es muy recomendable ya que no se pueden realizar transferencias binarias). La velocidad del puerto serie no tiene por que ser la misma que la de transmisión de los datos, de hecho debe ser superior. Por ejemplo, para transmisiones de 1200 baudios es recomendable usar 9600, y para 9600 baudios se pueden usar 38400 (o 19200).

Un ejemplo de conexión entre un PC y un equipo periférico con señal TTL



Sensores Analógicos y Digitales

Generalidades

Existe una gran variedad de sensores en el mercado de los cuales puedes disponer, claro, que antes habría que clasificarlos y aquí lo haremos brevemente...

Veamos... Los sensores pueden ser de dos tipos, analógicos y digitales.

Los sensores digitales son aquellos que frente a un estímulo pueden cambiar de estado ya sea de cero a uno o de uno a cero (hablando en términos de lógica digital) en este caso no existen estados intermedios y los valores de tensión que se obtienen son únicamente dos, 5V y 0V (o valores muy próximos)

Ahora bien, como los sensores comunmente serán utilizados con circuitos lógicos, y más si se trata de robótica en cuyo caso posiblemente incluyas un microcontrolador, habrá que ver como trabajar con los sensores analógicos. Por suerte existen unos Circuitos integrados llamados Conversores Analógico/Digital (Conversores A/D) que transforman la señal analógica en señal digital, y por supuesto también están los Conversores D/A, pero analicemos los primeros.

Introducción al conversores A/D (Analógico/Digital)

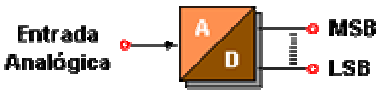
Aclaremos algo, esto no es una lección de conversores de este tipo, es sólo para que tengas una idea de ellos a grandes rasgos Ok.?, Sigamos... Los conversores Analógico/Digital los puedes seleccionar entre otras cosas, de acuerdo a la cantidad de bits de salida, por ejemplo...

Un Conversor A/D de 3 bits dispone de una entrada analógica y 3 terminales de salida Digitales, es decir que combinando las salidas puede tomar 8 posibles valores binarios sengún el nivel de tensión en su entrada. por aquello de 2^n es decir que tendrás valores entre 000 y 111, veamos como se corresponden estos valores con los niveles de tensión.

Entrada Analógica	Salida Digital
0 V	000
0.625 V	001
1.25 V	010
1.875 V	011
2.5 V	100
3.125 V	101
3.75 V	110
4.375 V	111

Te preguntarás ...que pasó con los 5V?, bueno, es que el conversor necesita un nivel de tensión para utilizarlo como referencia y en este caso utilicé los 5V, también podría ser 0V, o mejor aún ambos.

Aquí puedes ver una imagen representativa de un Conversor A/D, en ella se indican en la salida dos terminos muy utilizados MSB y LSB...



MSB es el valor binario más significativo y **LSB** es el menos significativo (en nuestro ejemplo, 111 y 000 respectivamente)

Lo visto hasta el momento te puede servir si en caso deseas decodificar una señal analógica y utilizarla como si fuera digital, por ejemplo en el caso de una fotocelda, esta varía su resistencia según la iluminación que recibe, por lo tanto es un sensor de tipo analógico. un pulsador tiene dos estados, activado o no, por lo tanto es de tipo digital.

Bueno, espero lo hayas comprendido...!!!

Aparte de aquello de los Conversores A/D, tienes la posibilidad de ingeniártelas para que una señal analógica tome dos estados y así solucionar tu problema, por ejemplo con una compuerta Schmitt Trigger (el CD40106 muy recomendado...!!!), las compuertas de este tipo tienen la ventaja de cambiar el estado de su salida en un determinado umbral de tensión de entrada, razón por la cual son muy utilizados para esta tarea, pero ya lo irás viendo en la descripción de cada tipo de sensor...

Creo que sería bueno que veas algunos temas que se pueden relacionar o aplicar a los sensores, consulta en la sección de IC Datos que hay algunos muy interesantes como el CD4013. Por ahora sólo un par de ellos luego iré agregando el resto.

Bueno, aquí dejo un par de sensores, su descripción y uno que otro circuito de prueba, repito... DE PRUEBA.

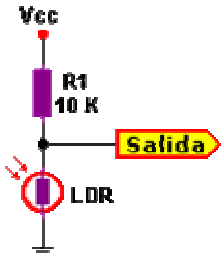
Sensores Analógicos

Fotocélulas, Fotorresistencias o LDR's

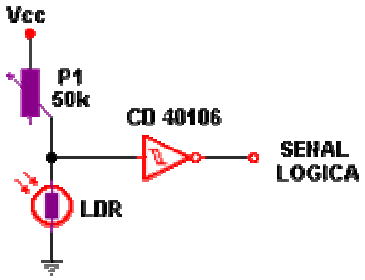
La LRD es quizás una de las más utilizadas en los sistemas sensores para robótica compiten a gran escala con los fototransistores.

Para comenzar debes saber que las LDR's son resistores que varían su valor de acuerdo a la intensidad de la luz, razón por la cual se trata de un sensor analógico, es decir que siempre toma valores distintos, no podrías tomar un valor lógico 1 o 0 como en lógica digital, pero nos la podemos arreglar con un par de trucos.

La fotocélula en total oscuridad puede llegar a tomar valores de 1M ...si no es más, y a plena iluminación a unos pocos k's o quizás menos. Lo que se puede hacer, es un arreglo entre la fotocelda al polo (-) y una resistencia fija al polo (+), de esa manera el punto de unión entre estos dos componentes podrá tomar dos valores según la variación de la LDR, señal que se puede utilizar como salida del sensor, este tipo de circuitos es conocido como divisor de tensión...



El tema es que la señal aun sigue siendo analógica, y para convertirla en señal digital podríamos utilizar un disparador Schmitt como el CD40106 que tiene 6 disparadores inversores en su interior, y nos quedaría averiguar las características de la fotocelda y la tensión de disparo del Schmitt y así seleccionar el nivel de tensión al que quieres trabajar, lo cual podrías hacerlo con un potenciómetro en lugar de la resistencia de 10k.

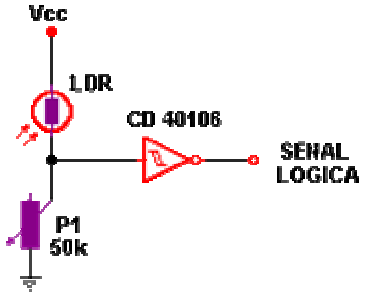


Con el potenciómetro P1 puedes seleccionar la sensibilidad a tu gusto, bueno, con alguna que otra limitación. Si deseas realizar los cálculos para averiguar la tensión en el punto medio, lo puedes hacer del siguiente modo...

V = LDR * (Vcc/(LDR+R1))

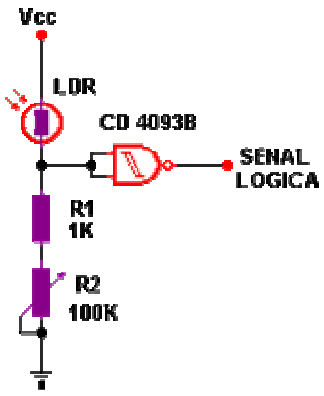
y así sabrás el nivel de tensión en distintas situaciones.

Esta no es la única forma, también puede darse el caso opuesto, observa...



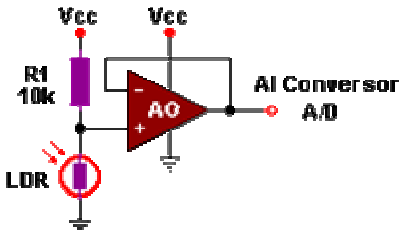
Todo depende de la forma en que deseas trabajar, en el caso anterior la señal lógica obtenida a plena iluminación es "0", mientras que en esta última es "1".

Ahora bien, Teniendo un poco de conocimiento de compuertas lógicas también puedes adoptar este circuito...



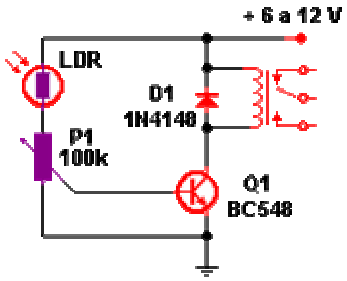
La lógica de funcionamiento es "1" a plena iluminación, aunque la puedes regular con R2.

Existe otra posibilidad aún más interesante y recomendada, se trata de utilizar un amplificador operacional con la intención de no afectar al divisor de tensión...



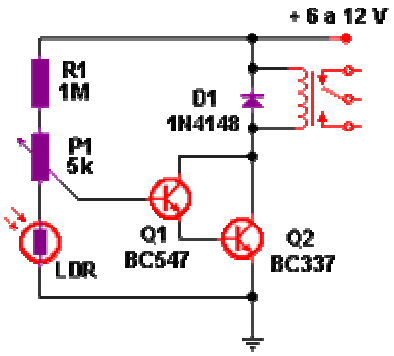
Aquí el AO. se conectó como seguidor de tensión, observa la realimentación negativa, esta conexión es conocido como configuración buffer, es decir, amplificamos un poco la señal para evitar pérdidas y así no obtener resultados inesperados, respecto al operacional utilizado bien puede ser el 741, el LM833 que es un doble operacional, o el LM324 que posee 4 operacionales en su interior, hay muchos de los cuales puedes elegir.

Todavía hay más, y es que puedes usar un transistor en corte y saturación para activar un relé por ejemplo, veamos eso...



En este caso, la salida del divisor de tensión está en el cursor del potenciómetro, al iluminar la fotocelda se alimenta la base del transistor y este pasa a plena saturación. La sensibilidad del circuito se ajusta con P1. Respecto al modo de configuración de transistores para que trabajen en modo corte y saturación puedes consultar la siguiente sección... ***Transistores***

No viene mal un ejemplo más, observa este circuito...



En este esquema puedes ver los transistores en dárlington, es con la idea de aumentar la ganancia del circuito y obtener un corte más profundo en el divisor de tensión, el tema es que con este último circuito el relé se mantiene inactivo siempre que haya iluminación en la LDR, y cuando esta iluminación se interrumpe se acciona el relé.

Bueno, espero que hasta aquí, todo haya sido de utilidad...

Emisión y Recepción de Infrarrojos

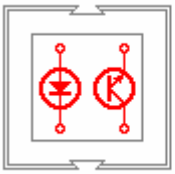
De los tipos de sensores que más llaman la atención, se destacan los sensores de luz, y entre ellos los conocidos Infrarrojos, y a ellos le dedicaremos este artículo...

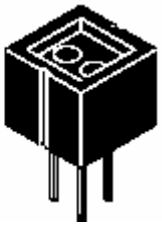
Para hacer una breve descripción de lo que es una radiación infrarroja, imagínate la luz del sol, pues esta contiene todas las bandas de colores con las cuales se compone la luz blanca (conocido como espectro de emisión de luz), ahora, en los extremos del espectro se encuentra la radiación infrarroja (IR) y la ultravioleta (UV), ambas son imposibles de ver, es decir son invisibles, pero están presentes y nosotros las vamos a poner en evidencia...

En microbótica tanto como en robótica, se hace uso de este tipo de emisión de luz (en especial la infrarroja) con la intención de detectar obstáculos sin que uno de nuestros modelos tome contacto físico con el mismo. En algunos casos una vez establecida la comunicación entre emisor y receptor, es posible realizar una transmisión de datos, sino imagínate... como hago para aumentar el volumen de mi TV, cambiar de canal, aumentar el brillo, quitarle el color..., etc., etc., y todo con un sólo emisor de IR...???

Pero bueno, sólo les mostraré lo básico, y cuando aprenda más les comentaré al respecto.

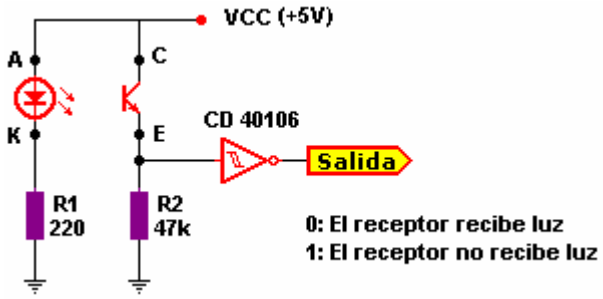
Existen encapsulados que traen incorporado en su interior tanto al emisor como receptor, de todos ellos, el más conocido es el CNY70, que cuenta con 4 pines, dos para el Diodo IR y dos para el fotoTransistor.





Pero, nosotros podemos conseguir algunos, en especial de las disqueteras viejas, en donde te encontrarás con algunos de estos tipos...

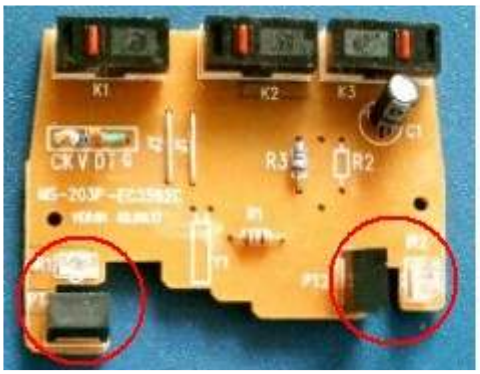
Para estos dispositivos podrías usar el siguiente circuito, que da buenos resultados, en la salida puedes agregarle un LED con su respectiva resistencia de 220R o 150R...

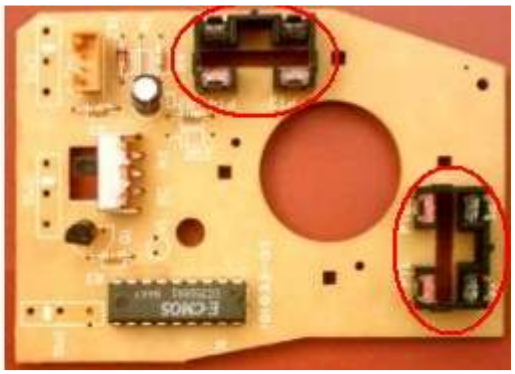


Vamos por otros, que hay más, pensaba comenzar con algún emisor de IR, pero antes necesitas un receptor, como emisor (en nuestras primeras pruebas) puedes utilizar el control de tu TV, pero no te preocupes que luego haremos el nuestro, jejejeje...

Los materiales que necesitaremos son muy fáciles de conseguir, busca un viejo mouse, el Control Remoto de tu TV, un integrado CD40106 y ya podemos comenzar...

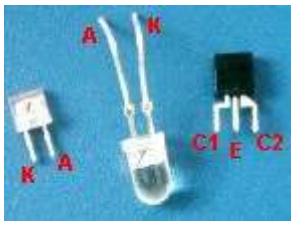
Al desmontar tu mouse, te encontrarás con 4 diodos IR y 4 fotodiodos, aunque en algunos te darás con 2 fototransistores (por lo general son negros) y sólo 2 IR, como los de la imagen... que más, un par de microswitchs, y una que otra cosita más, pero que ya no son de mucha importancia.





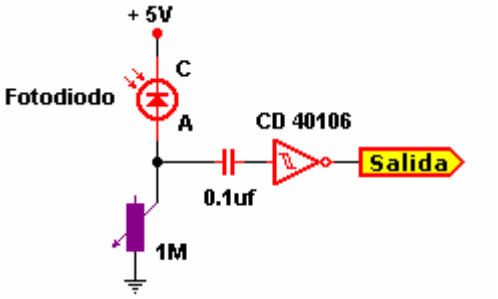
Ahora bien, cual es cual...???, en mi caso (de la segunda imagen) los que tienen una marca rosada son los fotodiodos, y los que tienen una marca negra los emisores IR, y en la primer foto, los blancos son los IR's y los negros los fototransistores.

Ok... Ahora nos toca desmontarlos con mucho cuidado, para no dañarlos... En la siguiente imagen, tienes los receptores de IR, nota que agregué uno más, el del medio, lo saqué de un viejo video-juego, y los que están de ambos lados son de los mouse's anteriores, notarás que el fototransistor negro tiene 3 terminales, de ellos, el de en medio es el emisor y los otros dos son los colectores del fototransistor, en realidad son dos fototransistores encapsulados en uno sólo con el emisor común, en otro tutorial veremos que uso le podemos dar, y hablaremos más de él...

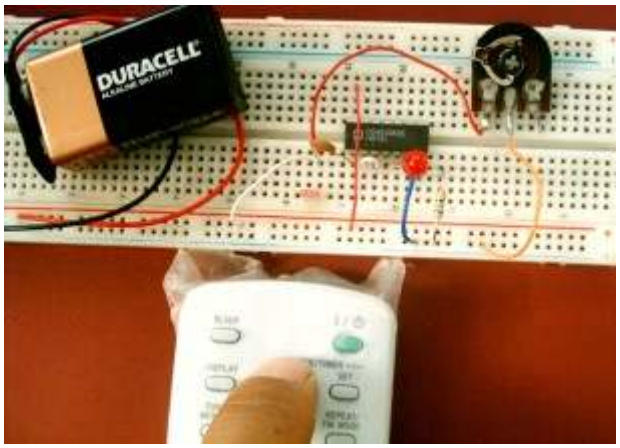


Antes de que empieces a preguntar, también tomé una imagen de los LED's emisores de IR, para que luego no tengamos problemas al hacer nuestros experimentos, bien, los dos pequeñines son de los Mouse's y el otro, lo conseguí de una casa de electrónica, que más, no podía ser de otra forma...

Ahora que ya disponemos de todos los materiales, podemos comenzar con nuestro primer circuito de prueba, que es el de un simple receptor, aquí tienes el esquema del circuito.



Este circuito te debe parecer familiar, y es que sí, lo vimos con los LDR, y no difiere mucho de los que ya conocemos, bien, este es el receptor, y para saber si realmente recibe la señal le colocaremos un LED, y enviaremos la señal con el Control del TV, así...

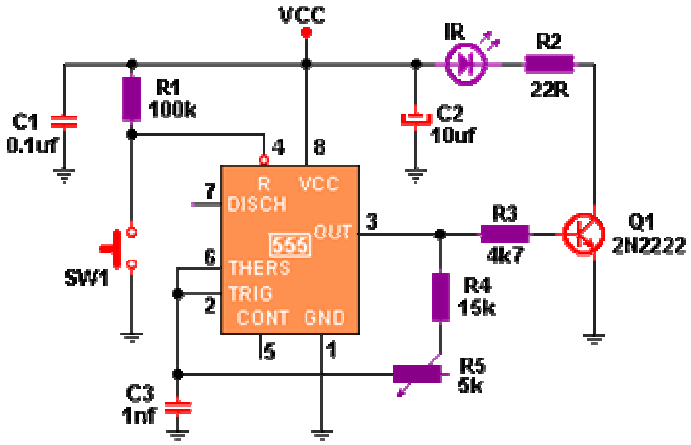


Creo que estamos yendo por buen camino, ya sabemos que realmente nuestro circuito funciona, pero hay algo muy curioso, resulta que en la prueba que acabamos de hacer, nuestro LED parpadea...

Lo que ocurre es que los controles emiten una señal codificada, y eso es lo que estamos viendo, es más la emisión debe tener una frecuencia aproximada a los 38 kHz (KiloHertz, es decir 38000 pulsos por segundo) que es la frecuencia que deben detectar la mayoría de los receptores o fotodiodos, como no voy a entrar en cálculos, les mostraré algunos de los circuitos que emiten esta frecuencia y que me dieron buenos resultados con estos diodos IR de los mouse's. Vamos por el primero...

El NE555.

Dicen, que entre los emisores, el mejor de todos es el que utiliza un circuito integrado NE555, y el que nunca me falló hasta ahora, es éste...



Observa que los pines 5 y 7 del integrado quedan libres. El interruptor que se encuentra en el circuito cumple la función de activarlo o desactivarlo. El preset o potenciómetro, permite regular la frecuencia de trabajo en un rango de 36 a 40 kHz (dependiendo del receptor utilizado). Reduciendo el valor de R3 puedes aumentar la intensidad de emisión y así su alcance. El transistor Q1 puede ser un 2N2222 o 2N2219, éste amplifica la corriente para el LED IR, y nos permitirá por ejemplo, utilizar otro LED más...

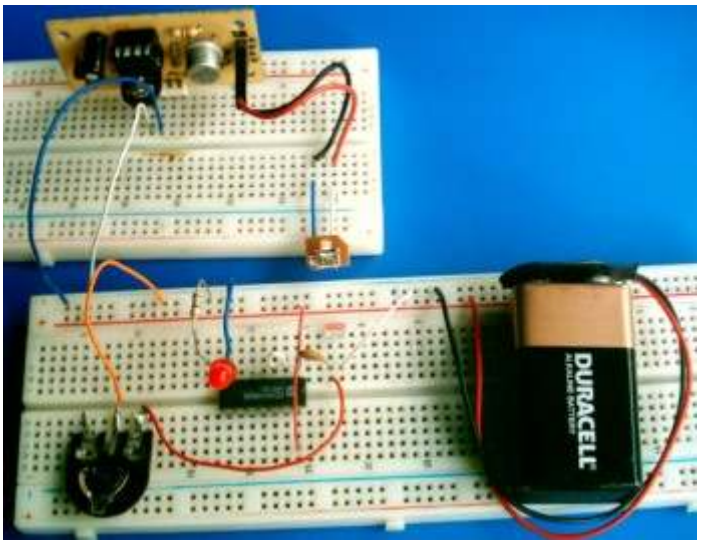
Luego de tantas pruebas, finalmente monté el circuito en una pequeña placa de 2,7cm x 5cm, aquí dupliqué IR y la resistencia R2 del circuito anterior, lo hice para conectar 2 Diodos IR, ya que creo que en otro momento me hará falta, y funciona...!!!

Ahora vamos a ver como se encienden estos IR, primero decirte que es imposible notarlos a simple vista, pero con la ayuda de una cámara digital verás que tiene un color violeta-lila-medio_blanco o bueno, algo así, mejor míralo...

IR Encendido

IR Apagado

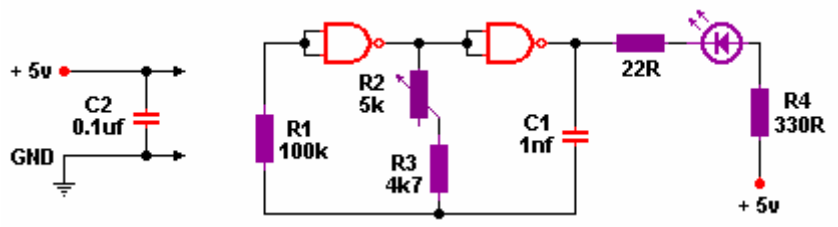
Ya estamos listos para hacer la prueba Emisor/Receptor y utilizaremos ambos circuitos montados en una placa de pruebas, y de paso vemos la distancia que alcanza...



A esto queríamos llegar, ya estarás viendo las aplicaciones que le podrás dar. Pero como no podía quedarme con esto, fui por más, pero esta vez lo haría con compuertas lógicas...

.: El CD4011.

Este integrado contiene 4 operadores NAND en su interior, de los cuales dos serán utilizados para hacer un multivibrador que cumpla con las características indicadas, el esquema del circuito es el que sigue...

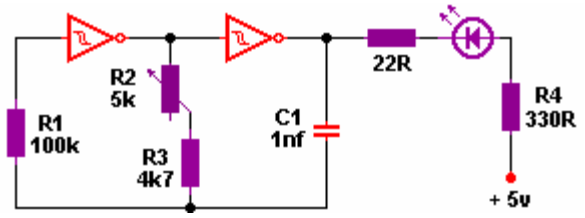


Se debe establecer un pequeño filtro en la fuente de alimentación de todo el circuito, es por eso que se agregó un capacitor de 0.1uf. Puedes disminuir el valor de R4 para darle mayor intensidad al IR. igual que antes, con R2 puedes regular la frecuencia del circuito.

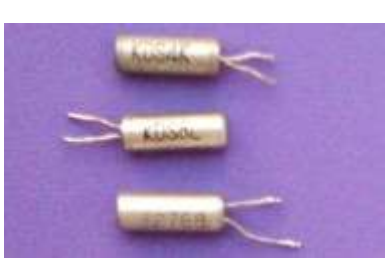
Ahora bien, ya habíamos visto algo de estos multivibradores en el tutorial de electrónica digital, de hecho, unir las entradas de estos operadores, me hace recordar que también podría hacerlo con simples inversores, o mejor aún, con compuertas tipo Schmith Trigger...

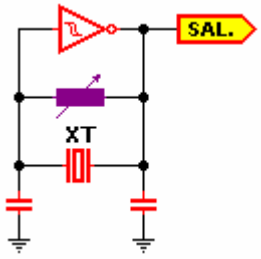
El CD40106.

Este Integrado posee 6 compuertas inversoras, del tipo Schmith Trigger, de las cuales utilizaremos sólo 2, y montaremos un circuito similar al anterior, es más el circuito es el mismo, sólo cambian los operadores utilizados, observa...

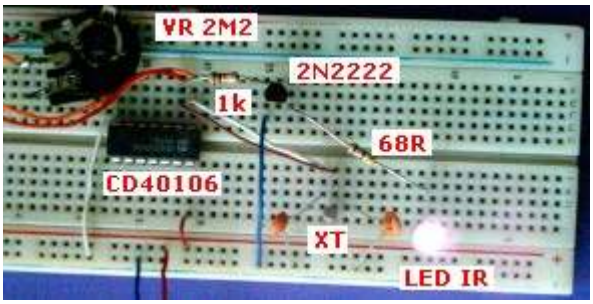


Si bien los mouse's traen sus emisores infrarrojos, es de suponer que alcanzan la frecuencia indicada, pero para mayor sorpresa, es que traen en su circuito, un Cristal de Cuarzo cilíndrico que suele ser de 4 Khz, como el de la siguiente imagen, y si recapitulamos los multivibradores con compuertas lógicas y Cristales de Cuarzo... llegaremos al siguiente circuito...

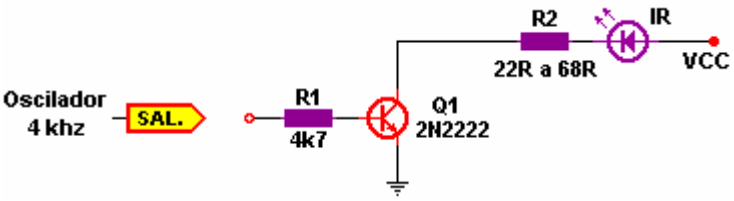




Que también, obviamente, funciona perfecto. Aquí, los capacitores que van al cristal son de 27 pf y en la práctica, utilicé un preset (resistencia variable) de 2,2M.



En todos los casos, es bueno hacer estas prácticas utilizando en la salida de los osciladores un transistor como el 2N2222, que es un transistor de batalla, el circuito para este transistor y el IR lo puedes ver en la siguiente imagen...

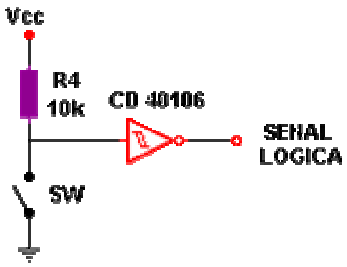


Bueno, llegamos al final, sólo decirte que todos estos circuitos dieron buenos resultados, al menos a mí, espero sea de utilidad, y de ayuda para que armes un circuito mucho más interesante que simples emisores, te imaginas... podrías armarte de una placa detectora de obstáculos, que sea pequeña, sencilla de montar, y que sirva para cualquier propósito, como para hacer un seguidor de líneas o algo así...???, estaría bueno no crees...???

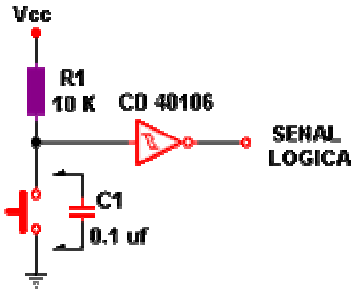
Sensores Digitales

Switchs, Interruptores y Microswitchs, o Bumpers

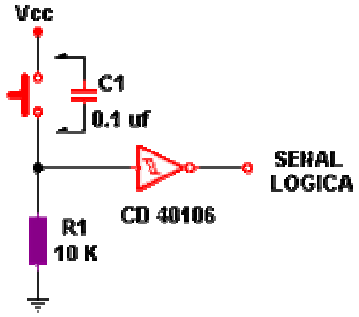
Aunque parezca mentira, los Switchs son muy utilizados como dispositivos sensores, por ejemplo, si deseas que un modelo realice una determinada acción cuando choque con algún obstáculo recurres a ellos, al margen del tipo de interruptor que quieras utilizar, el circuito básico será siempre el mismo, UN DIVISOR DE TENSIÓN...



A pesar de que los interruptores son sensores de tipo lógico (por trabajar con niveles 0 y 1) es mejor acondicionar los niveles de tensión para ellos, es por eso que incluí el CD40106. Debo aclarar que el circuito anterior presenta un pequeño inconveniente, y es que al activarse se pueden producir rebotes eléctricos, es decir, cuando crees haber enviado un 1 lógico en realidad enviaste varios, es como si se los hubiera presionado varias veces, pero calma...!!! que todo tiene solución...

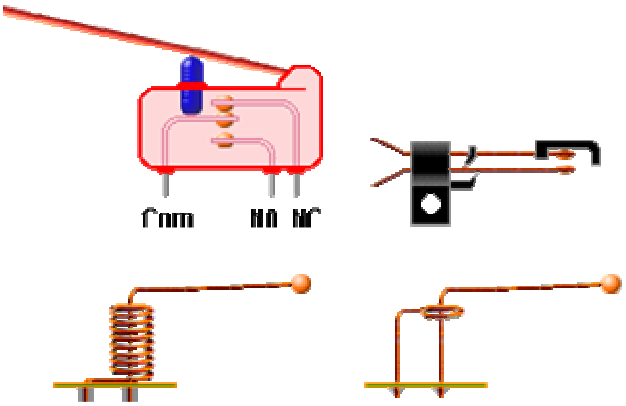


Agregándole un pequeño capacitor, como los de 0.1uf puedes evitar esos rebotes. Claro que según el tipo de señal que tu modelo necesite, ya sea 0 o 1 te servirá el circuito anterior o este...



En fin, el tema es que hay muchos modelos de este interruptor, pero los más utilizados en microbótica son los Bumpers, ya sean comerciales o que más da, los tuyos propios y originales :o))

Veamos algunos ejemplos...



Creo que fue suficiente, Luego los aplicaremos a algún modelo específico, esto sólo fue para que tuvieras una idea. Los dos primeros son comerciales, y los dos de abajo son arreglos para implementarlos como sensores tipo bigote de gato. Me pareció bueno incluirlo así que, ahí está...

Descripción y uso de algúnos Circuitos Integrados (IC's)

Breve Introducción

En muchas ocasiones es bueno saber para que se utilizan algunos Componentes o Circuitos Integrados, o bien, conocer la forma de utilizarlos adecuadamente para así sacarle el mayor partido, es por eso que dejé este espacio para dedicárselo a ellos.

Toda la información aquí expuesta está extraída de manuales, libros, revistas y la web con su valioso aporte, pero quiero aclarar algo, las funciones de cada uno de ellos según lo indicado en estas páginas fueron personalmente probadas y verificadas en mi pequeño laboratorio.

Si por algún motivo contigo no funcionan es por que el integrado o componente en cuestión se encuentra en malas condiciones o seguramente metiste la pata en el lugar equivocado, o simplemente es cuestión de un falso contacto, lo cual es más frecuente, o bueno, en fin, quién sabe...!!! Razón por la cual te recomiendo que verifiques eso justamente.

Empezaré por unos pocos, sin un orden cronológico ni nada por el estilo, quizás luego lo haga, por ahora sólo los elegí al azar...

Por cierto, y antes de que lo olvide en la sección de Software se describe un programa con el que puedes simular la funcionalidad de cada uno de estos componentes.

NE 555
Es un Timer de precisión, pero pronto se le encontraron otra aplicaciones tales como osciladores astables, generadores de rampas, temporizadores secuenciales, etc.,

CD 4013
Es un doble flip-flop tipo D, con muchas aplicaciones interesantes como; monoestable, llave osccilante, divisor por dos, etc.

CD 4017
Es un contador/divisor o decorificador con 10 salidas, puede dividir o contar por cualquier valor entre 2 y 9, con recursos para continuar o detenerse al final del ciclo.

CD 4029
Es un contador sincrónico "Up-Down" de 4 bits, Progresivo/Regresivo, Preajustable, Binario/Década.

CD 4066
Es un cuádruple interruptor Bilateral, diseñado para la transmisión o multiplexado de señales digitales o analógicas, cada interruptor dispone de un pin de control y dos de entrada/salida...

CD 4511
Se trata de un excitador/decodificador/cerrojo BCD a 7 Segmentos. El circuito provee las funciones de un cerrojo (latch) de almacenamiento de 4 bit, un decodificador BCD 8421 a 7 segmentos, y gran capacidad de excitación de salida.

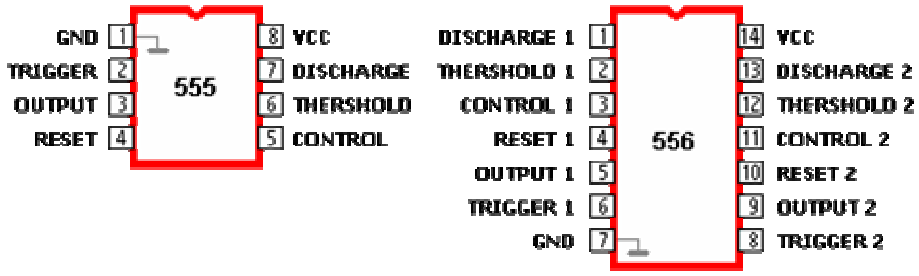
Circuito Integrado NE555

Presentación

Es uno de los Circuitos Integrados más famosos, de los más utilizados. Según el tipo de fabricante recibe una designación distinta tal como TLC555, LMC555, uA555, NE555C, MC1455, NE555, LM555, etc. aunque se lo conoce como **"el 555"** y ya todos saben de que se está hablando.

Respecto al formato o encapsulado, puede ser circular metálico, hasta los SMD, pasando por los DIL de 8 y 14 patillas.

Existen versiones de bajo consumo con el mismo patillaje y versiones dobles, es decir que contienen 2 circuitos iguales en su interior, que comparten los terminales de alimentación y se conocen con la designación genérica de 556, observa la siguiente imagen...



Utilización:

Este circuito es un "Timer de precisión", en sus orígenes se presentó como un circuito de retardos de precisión, pero pronto se le encontraron otra aplicaciones tales como osciladores astables, generadores de rampas, temporizadores secuenciales, etc., consiguiéndose unas temporizaciones muy estables frente a variaciones de tensión de alimentación y de temperatura.

Características generales:

El circuito puede alimentarse con tensión continua comprendida entre 5 y 15 voltios, aunque hay versiones que admiten tensiones de alimentación hasta 2 V., pero no son de uso corriente. Si se alimenta a 5V es compatible con la familia TTL.

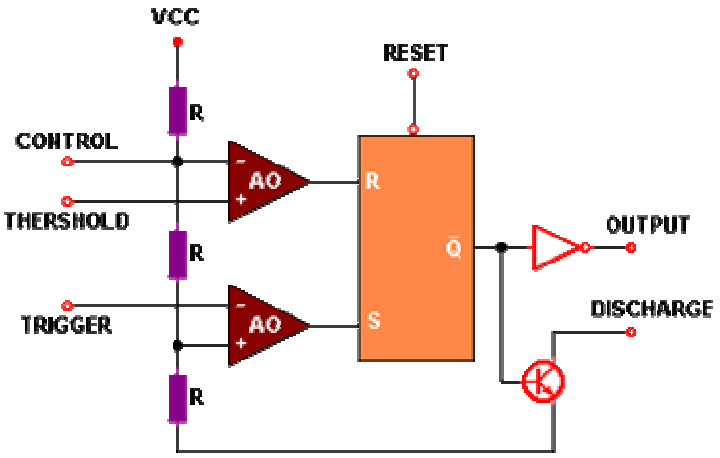
La corriente de salida máxima puede ser de hasta 200mA., muy elevada para un circuito integrado, permitiendo excitar directamente relés y otros circuitos de alto consumo sin necesidad de utilizar componentes adicionales. La estabilidad en frecuencia es de 0,005% por ºC.

Necesita un número mínimo de componentes exteriores, la frecuencia de oscilación se controla con dos resistencias y un condensador. Cuando funciona como monoestable el retardo se determina con los valores de una resistencia y de un condensador.

.: Diagrama de Bloques Interno:

El funcionamiento y las posibilidades de este circuito se pueden comprender estudiando el diagrama de bloques. Básicamente se compone de dos amplificadores operacionales montados

como comparadores, un circuito biestable del tipo RS del que se utiliza su salida negada, un buffer de salida inversor que puede entregar o absorber una corriente de 200mA. y un transistor que se utiliza para descarga del condensador de temporización.



Una red de tres resistencias iguales fija los niveles de referencia en la entrada inversora del primer operacional, y en la no inversora del segundo operacional, a 2/3 y 1/3 respectivamente de la tensión de alimentación.

Cuando la tensión en el terminal umbral (THRESHOLD) supera los 2/3 de la tensión de alimentación, su salida pasa a nivel lógico "1", que se aplica a la entrada R del biestable, con lo cual su salida negada, la utilizada en este caso, pasa a nivel "1", saturando el transistor y comenzando la descarga del condensador, al mismo tiempo, la salida del 555 pasa a nivel "0".

Pasemos ahora al otro amplificador operacional, si la tensión aplicada a la entrada inversora, terminal de disparo (TRIGGER), desciende por debajo de 1/3 de la tensión de alimentación, la salida de este operacional pasa a nivel alto, que se aplica al terminal de entrada S del biestable RS, con lo que su salida se pone a nivel bajo, el transisor de descarga deja de conducir y la salida del 555 pasa a nivel lógico alto.

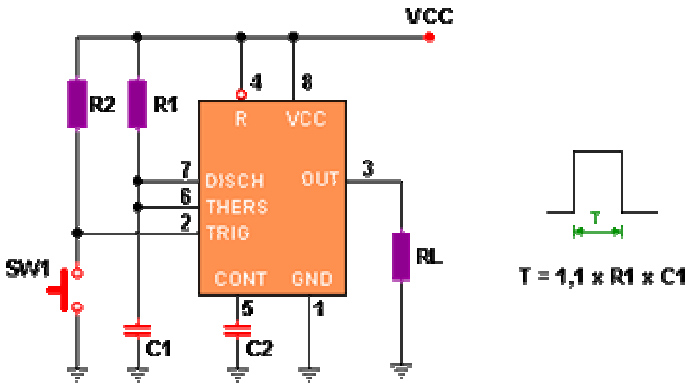
La gama de aplicaciones del circuito se incrementa, pues se dispone de un terminal de reset, activo a nivel bajo, que se puede utilizar para poner a nivel bajo la salida del 555 en cualquier momento.

Algunas de sus aplicaciones

Circuito monoestable:

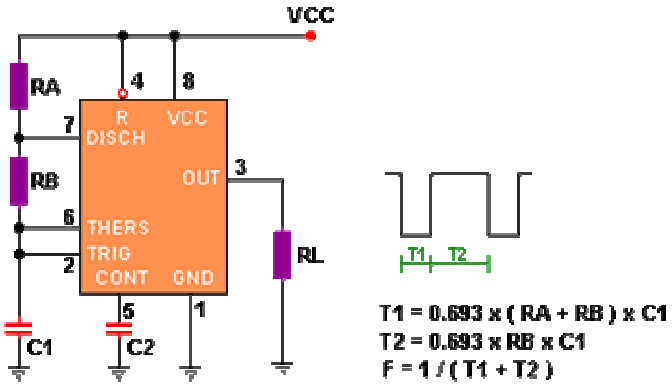
La salida del circuito es inicialmente cero, el transistor está saturado y no permite la carga del condensador C1. Pero al pulsar SW1 se aplica una tensión baja en el terminal de disparo TRIGGER, que hace que el biestable RS cambie y en la salida aparezca un nivel alto. El transistor deja de conducir y permite que el condensador C1 se cargue a través de la resistencia R1. Cuando la tensión en el condensador supera los 2/3 de la tensión de alimentación, el biestable cambia de estado y la salida vuelve a nivel cero.

R2 esta entre 1k y 3,3 M, el valor mínimo de C1 es de 500pf.



Circuito astable:

Cuando se conecta la alimentación, el condensador está descargando y la salida del 555 pasa a nivel alto hasta que el condensador, que se va cargando, alcanza los 2/3 de la tensión de alimentación, con esto la salida del biestable RS pasa a nivel "1", y la salida del 555 a ceroy y el condensador C1 comienza a descargarse a través de la resistencia RB.

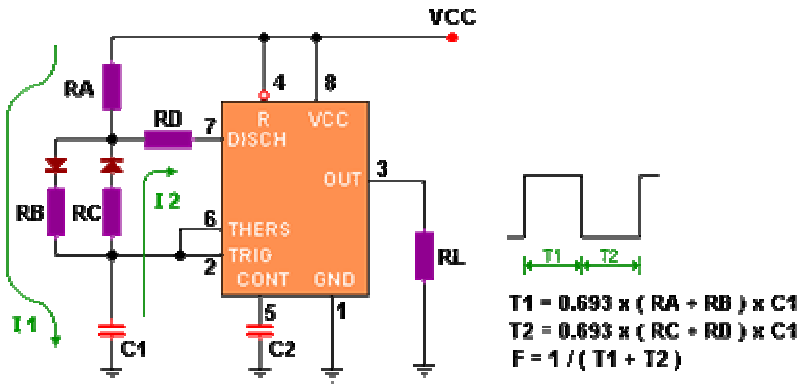


Cuando la tensión en el condensador C1 llega a 1/3 de la alimentación, comienza de nuevo a cargarse, y asi sucesivamente mientras se mantenga la alimentación.

RA toma valores entre 1k y 10M, RB<RA

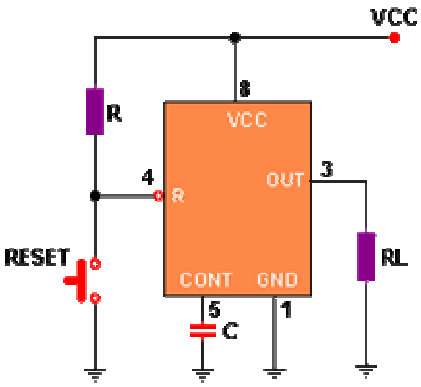
Circuito astable con onda simétrica:

En este circuito astable se muestra cómo puede obtenerse una onda simétrica; el modo de hacerlo es que el condensador tarde el mismo tiempo en cargarse que en descargarse, los caminos de carga y descarga deben ser iguales y se separan con dos diodos. El condensador C2 evita fluctuaciones de tensión en la entrada de control.



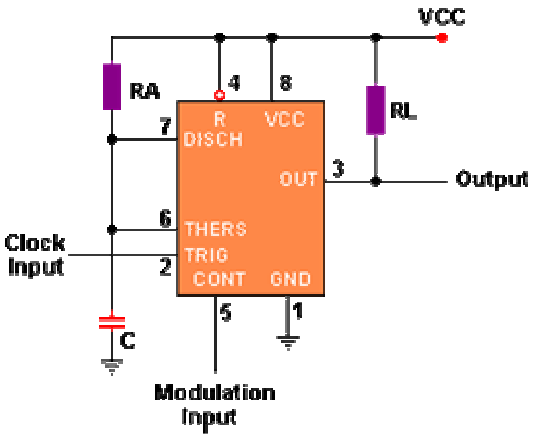
Terminal de Reset:

El terminal de reset puede conectarse directamente al positivo o bien mantener el nivel alto por medio de una resistencia, por ejemplo de 2k2. Al actuar sobre el pulsador, la salida del 555 pasa a nivel bajo directamente. Es como poner el integrado en un estado de reposo.



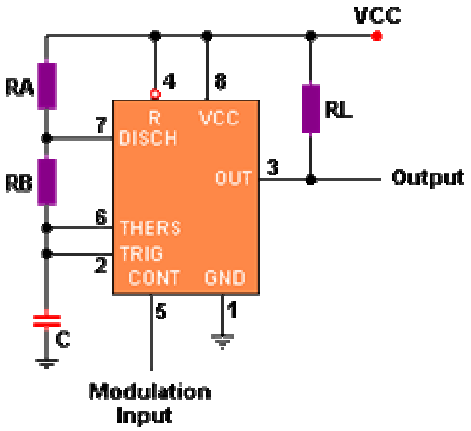
Modulación del ancho de pulso:

Aplicando una señal de nivel variable a la entrada de CONTROL el pulso de salida aumenta de ancho al aumentar el nivel de esa tensión.



Modulación del retardo de pulso:

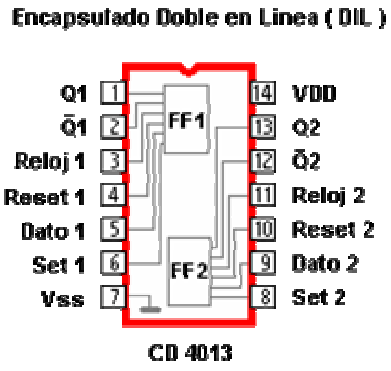
Aquí el pulso de salida aparece con mayor o menor retardo según aumente o disminuya la tensión aplicada al terminal de control.



Circuito Integrado CD4013

Descripción

Este integrado es un doble flip-flop tipo D, en la figura siguiente se puede observar la disposición de terminales y la tabla de verdad correspondiente, numerada de 1 a 6 para interpretar con mayor claridad el análisis de la misma...



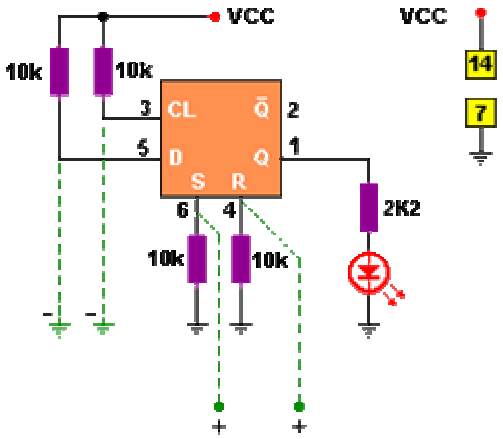
	CL†	D	R	S	Q	Q̄
1	↑	0	0	0	0	1
2	↑	1	0	0	1	0
3	↓	x	0	0	Q	Q̄
4	x	x	1	0	0	1
5	x	x	0	1	1	0
6	x	x	1	1	1	1

† = Cambio de nivel
x = No Importa

- **1 :** En este renglón las entradas están todas en "0"; la transición en sentido positivo del pulso de reloj, no tiene efecto en las salidas, por lo que la salida Q se mantiene en 0 y -Q en 1.
- **2 :** Con las entradas Set y Reset a potencial 0 y el dato a 1, si en la entrada reloj se presente un pulso de transición positiva el Flip-Flop cambia de estado y se mantiene en él, aun después de desaparecer dicho pulso.
- **3 :** Si el pulso de reloj es de transición negativa, aunque las entradas Set y Reset estén a 0, no conmutará independientemente del nivel de la entrada Dato, que puede ser 1 o 0, ya que sólo lo hace en la transición positiva.
- **4 :** En este caso x en la entrada de Reloj y Dato significan que es irrelevante el nivel que tengan ya que al estar a 1 la entrada Reset, el Flip-Flop no producirá ningún cambio.
- **5 :** No tiene importancia la polaridad de las entradas de Reloj y Dato, ya que el cambio de estado se produce llevando Set a 1, y se mantendrá en él aunque esta entrada vuelva a 0. Sólo se volverá al estado anterior (reposo) llevando momentáneamente la entrada Reset a 1.
- **6 :** Esta es una situación en la cual continúa funcionando como R-S, pero con la particularidad de ser seguidor de la señal presente en la entrada Set. Sigue sin tener importancia los niveles de Reloj y Dato. Al llevar el Set a 1, la salida -Q cambia también a 1, pero no lo hace la Salida Q, con lo que no se obtienen los estados complementarios; la salida Q se mantendrá a 1 todo el tiempo que esté a 1 la entrada Set, en cuanto esta entrada vuelva a 0 la salida Q también volverá a 0, esto es así porque la entrada Reset está a nivel 1, y como ya sabemos con positivo en este terminal el Flip-Flop se mantiene en estado de reposo.

Circuitos prácticos:

A continuación se describe un circuito general que verifica el funcionamiento del Flip-Flop según su tabla de verdad.



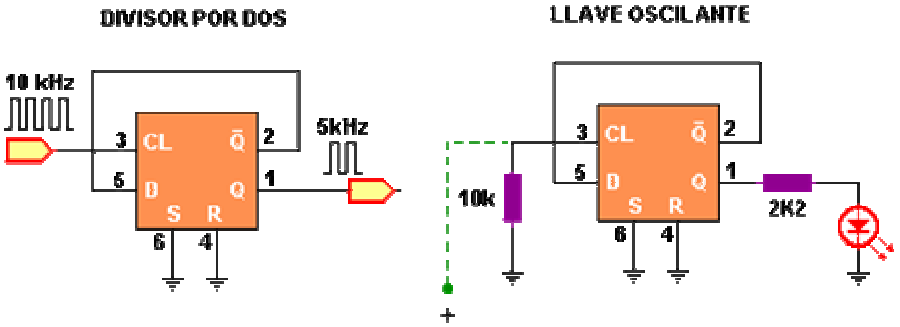
Los cables de prueba son los que estas con líneas de puntos. Los pulsos de la entrada de reloj (CL) se simulan mediante la conexión y desconexión del cable de prueba del terminal 3, (Masa (0) - Flanco de descenso, Vcc (1) - flanco de ascenso).

El LED solo brillará ante la conmutación o puesta a uno del Flip-Flop, puedes experimentar reseteando el circuito (con un 1 en Reset). Un detalle a tener en cuenta es que se pueden producir ruidos eléctricos (rebotes) al conectar los cables de prueba, pero puedes utilizar un capacitor para solucionar este inconveniente.

Contador Divisor por dos y/o Llave Oscilante I:

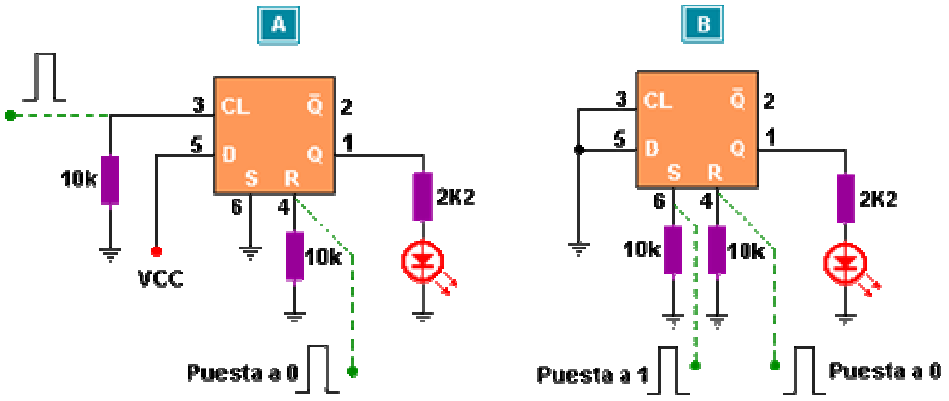
En este caso a modo de ejemplo se estableció 10kHz, con lo que se obtiene a la salida Q : 5kHz.

La llave oscilante o vaivén, cambia de estado con cada pulso de flanco ascendente en la entrada Reloj, en el primero se pone a uno, en el segundo se pone a cero.



Contador Divisor por dos y/o Llave Oscilante II:

Se obtiene el mismo efecto que en el caso anterior pero con dos señales de control una para la puesta a 1 y otra de las mismas características para la puesta a 0.

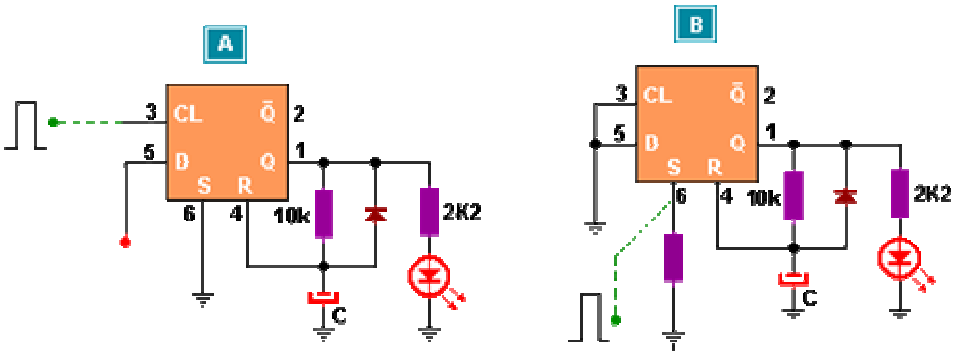


En **"A"** se trabaja para la puesta a 1 con la entrada de Reloj, y para la puesta a 0 con Reset. Se requiere que la entrada Dato se encuentre en 1.

En **"B"** se utiliza como un clásico FF R-S, es decir la entrada Set para la puesta a 1 y la entrada Reset para la puesta a 0. Las entradas de Reloj y Dato, en este caso son irrelevantes, es como si no existieran, por lo tanto no tiene importancia el nivel en ellas, en este caso se conectaron a masa para no dejarlas al aire, si estarían a VCC daría igual.

Flip-Flop Conectado como Monoestable:

Para realizar un monoestable con este integrado solo se requiere de una Resistencia, un Diodo y un Capacitor. También en este caso se puede operar desde las entradas Reloj o Set y se obtendrán los mismos resultados.



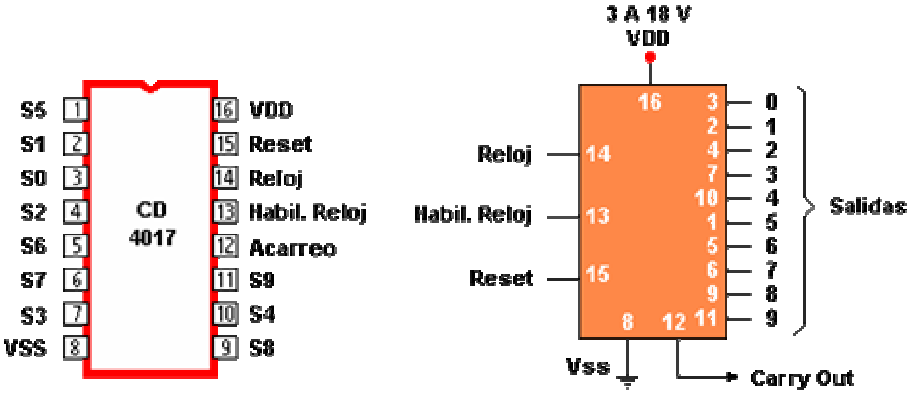
En **"A"** es operado con la entrada Reloj - En **"B"** es operado desde la entrada Set.

En ambos casos un pulso de transición positiva produce el cambio o puesta a 1, con lo cual la salida Q pasa al estado alto. A partir de este instante, el capacitor conectado en Reset comienza a cargarse. Cuando la magnitud de la tensión de carga supera el estado intermedio, actúa la entrada Reset y produce un nuevo cambio que lo pone en estado de reposo nuevamente. La duración del pulso de salida lo determina la red R-C. El diodo se conecta para sacarlo rápidamente del estado intermedio una vez producida la vuelta a 0. Durante la carga del capacitor se encuentra en oposición a la corriente, pero una vez que la salida Q vuelve a 0, queda en directo para la carga obtenida y lleva este potencial a 0 casi instantáneamente.

Circuito Integrado CD4017

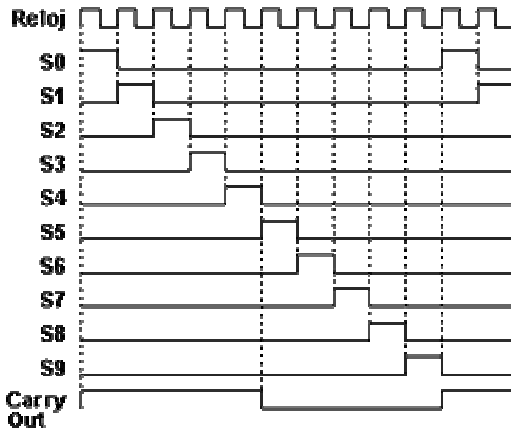
Descripción

Se trata de un contador/divisor o decorificador con 10 salidas. Estructuralmente está formado por un contador Johnson de 5 etapas que puede dividir o contar por cualquier valor entre 2 y 9, con recursos para continuar o detenerse al final del ciclo.



El Diagrama de funciones:

Para comprender mejor su funcionamiento lo haremos utilizando el diagrama de funciones.



Con las entradas "Habil. Reloj" y "Reset" a tierra, el contador avanza una etapa a cada transición positiva de la señal de entrada (Reloj). Partiendo entonces de la situación inicial en que "S0" se encuentra a nivel alto y todas las demás a nivel bajo. Con la llegada del primer pulso de entrada tenemos la primera transición. "S0" pasa a nivel bajo y "S1" a nivel alto, todas las demás permanecen en cero.

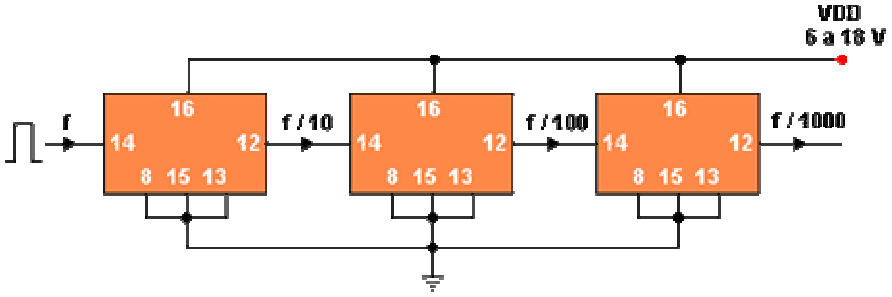
Con el segundo pulso, "S1" pasa a nivel bajo y "S2" a nivel alto, y así sucesivamente hasta la última.

Los otros terminales:

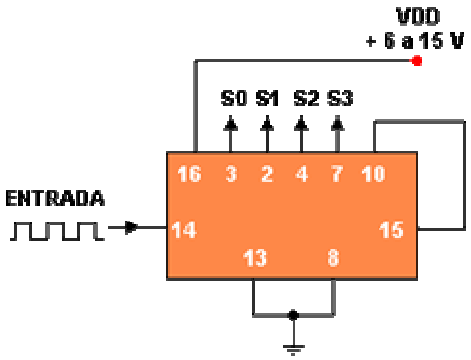
"Habil. Reloj" si está a tierra, hará que se inicie un nuevo ciclo. si está a VDD se consigue solo

un ciclo de funcionamiento.

"Carry-Out" Este terminal proporciona un ciclo completo a cada 10 pulsos de entrada, pudiendo usarse para excitar otro 4017 para división sucesiva de frecuencia o recuento por un número superior a 10



"Reset" Si se le aplica un nivel alto, lleva ese nivel al terminal "S0", volviendo a iniciar el recuento. Eso significa que si conectamos este terminal a cualquier salida, cuando ésta se lleve a nivel alto se iniciará un nuevo ciclo. Es decir que si conectamos "S4" a la entrada "Reset" tendremos un recuento sólo hasta 4.

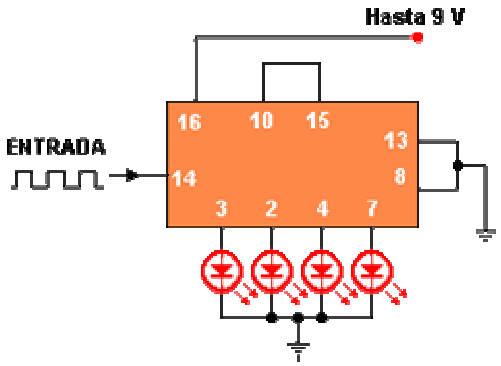


Circuitos de prueba:

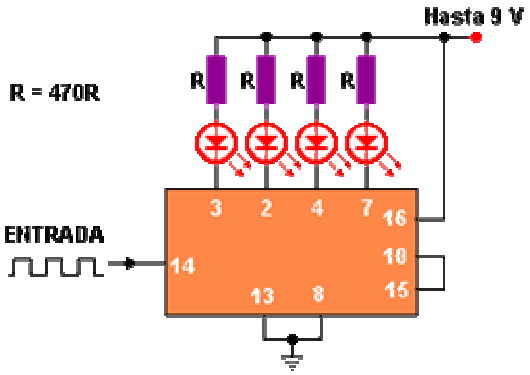
Las salidas de este integrado proporcionan corrientes lo suficientemente intensas como para excitar LED's y en aplicaciones de mayor potencia, transistores comunes.

Los leds pueden conectarse de dos maneras:

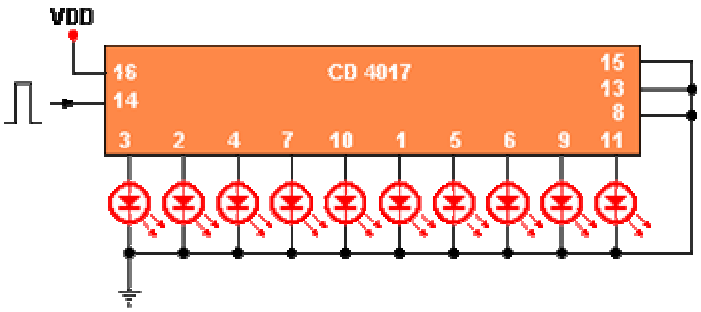
- **En modo positivo :** Solo el LED que tiene el nivel alto permanece encendido.



- **En modo negativo:** Solo el LED que tiene el nivel alto permanece apagado.



también puedes realizar una secuencia completa con todas las salidas, algo así como lo que se ve en la siguiente imagen



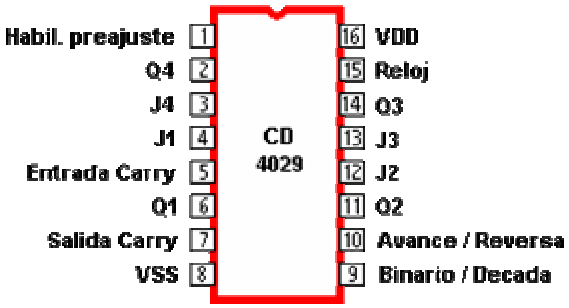
Circuito Integrado CD4029

Descripción

El CD4029 es un contador sincrónico "Up-Down", Contador Progresivo/Regresivo, Preajustable, Binario/Década

Es un contador de 4 bits que opera como década (0000 a 1001), tanto en el sentido creciente como decreciente.

Además de la entrada normal de clock (información de tipo serie) posee 4 entradas disponibles para información paralela. Esas entradas permiten que el contador sea cargado con un determinado dato (número binario) e inicie a partir de allí la cuenta.



En esta imagen tenemos la disposición de los pines del CD4029, y a continuación, una relación de todas sus entradas y salidas, con sus respectivas funciones.

Habilitación Preajuste (Pin 1): Entrada para lectura paralela. Cuando ampliamos un nivel lógico "1", en esta entrada, el contador se carga con la información presente en las entradas paralelas; si no es utilizada debe ser mantenida en "0".

J1, J2, J3, J4 (Pin's 3, 4, 12 y 13 respectivamente): Entradas paralelas de datos. Estas entradas actúan directamente en las salidas del contador, independientemente de cualesquiera otras condiciones, una vez que representan los "presets" de cada flip-flop del contador. En este circuito el digito más significativo (con peso 8) es alterado a través de la entrada J4 (pin3). Si quisieras por ejemplo, que el contador vaya al número 6, debes aplicar a las entradas paralelas el número correspondiente en binario, o sea 0110 (J4=0, J3=1, J2=1, J1=0,).

Q1, Q2, Q3 y Q4 (Pin's 6, 11, 14 y 2 respectivamente): Salidas del contador. La salida Q4 representa el dígito más significativo del número en binario, por lo tanto Q1 es el menos significativo.

Reloj (Pin 15): Entrada de Reloj. A cada transición ascendente (de "0" a "1") de la señal de Reloj el contador cambia de estado.

Avance/Retroceso (Pin 10): Entrada de control para cuentas crecientes o decrecientes. Un nivel lógico "1" hace que el circuito realice la cuenta creciente, con un nivel "0" la cuenta será decreciente.

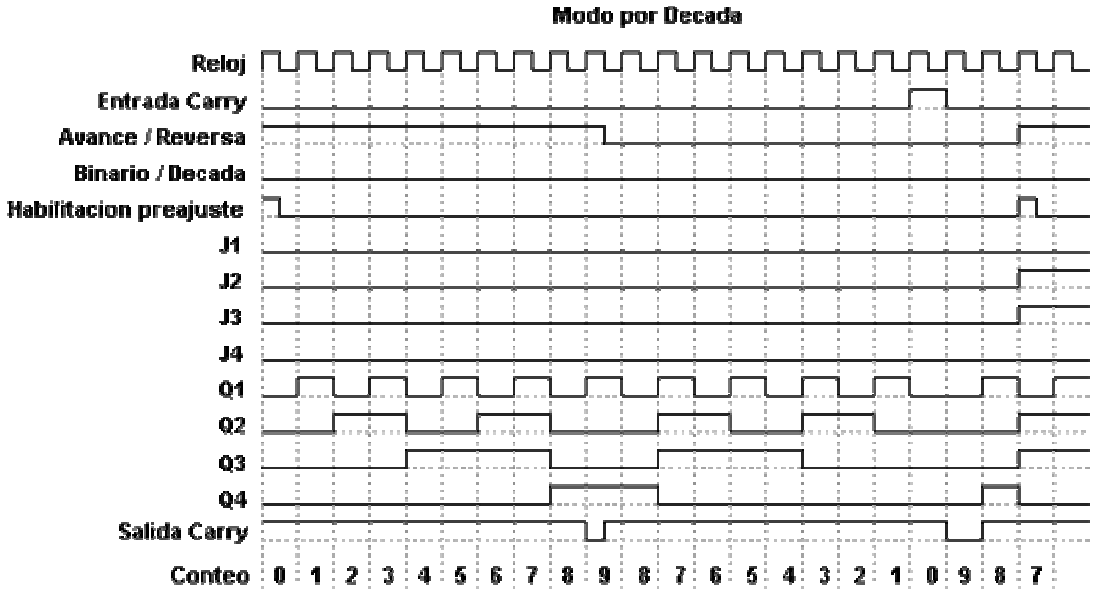
Binario/Década (Pin 9): Entrada de control para la cuenta en binario o década/decena. En

nivel lógico "1" el circuito cuenta en binario (de 0000 hasta 1111) y en nivel "0" el contador se comporta como una década (contando de 0000 hasta 1001 o sea de 0 a 9).

Entrada Carry (Pin 5): Habilitación del contador Con "1" la cuenta se paraliza. Con "0" el contador funciona normalmente.

Salida Carry (Pin 7): Salida de término de cuenta. Representa la salida "Carry" o "va 1" del contador. El nivel lógico de esta salida varía de "1" a "0" toda vez que el contador alcanza el número máximo de la cuenta, cuando está conectado como contador creciente, o cuando alcanza el menor número de la cuenta, al funcionar como contador decreciente.

En las siguientes imágenes se describe el diagrama de funcionamiento en **Modo Década** y en **Modo Binario**



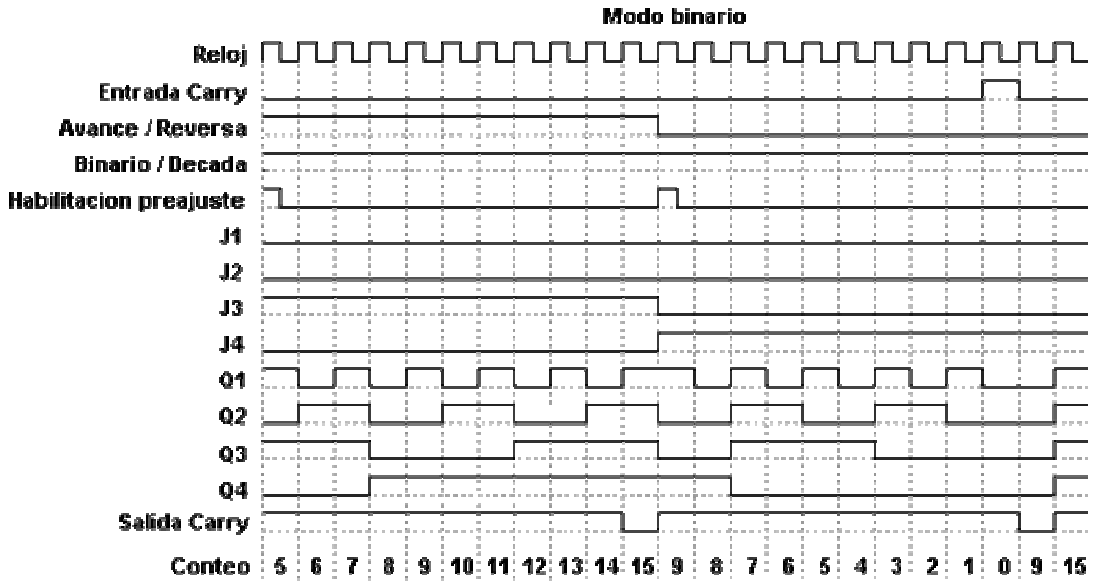


Tabla de Verdad:

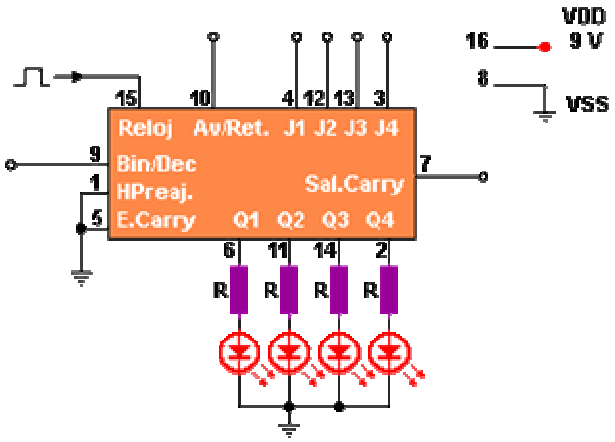
En la siguiente tabla de verdad se muestran los posibles modos de operación y sus

correspondientes entradas de control. En esta tabla los estados representados por "x" son irrelevantes, o sea, pueden ser "1" o "0", indistintamente.

RELOJ	HPreaj.	E.Carry	BIN/DEC	Av/Ret.	MODO DE OPERACION
X	1	X	X	X	Lectura de las entradas J1 a J4
X	0	1	X	X	Cuenta paralizada
↓	0	0	0	0	Cuenta decreciente en decada
↓	0	0	0	1	Cuenta creciente en decada
↓	0	0	1	0	Cuenta decreciente en binario
↓	0	0	1	1	Cuenta creciente en binario

Circuito de Prueba:

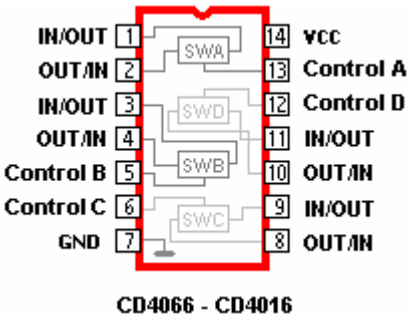
Para verificar la tabla de verdad citada anteriormente puedes utilizar el siguiente circuito de ejemplo, las resistencias son todas de 220 ohm.



Circuito Integrado CD4066

Presentación

El integrado en sí, es un cuádruple interruptor Bilateral, diseñado para la transmisión o multiplexado de señales digitales o analógicas, puedes considerarlo como un interruptor de conmutación, cada uno de los interruptores dispone de un pin de control y dos pines de entrada/salida, una representación de lo que se vería por dentro sería algo así...



CD4066 - CD4016

Como verás, también incluí otro integrado en el mismo esquema ya que son compatibles pin a pin, es decir puedes sustituirlo sin ningún inconveniente.

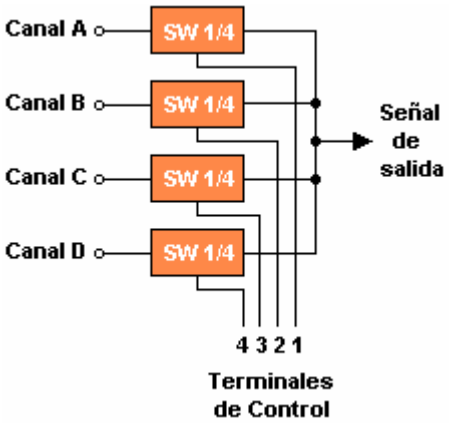
Que sea bidireccional significa que cualquiera de los dos pines de cada interruptor exceptuando al pin de control, puede hacer de entrada mientras el otro es de salida.

Cada interruptor entra en conducción cuando se presenta un nivel alto (superior al 70% de VCC), y en corte cuando se envíe un nivel bajo (inferior al 30% de VCC) por el mismo pin de control.

Veamos ahora un par de aplicaciones en las cuales puedes utilizarlo...

Selector de señales analógicas:

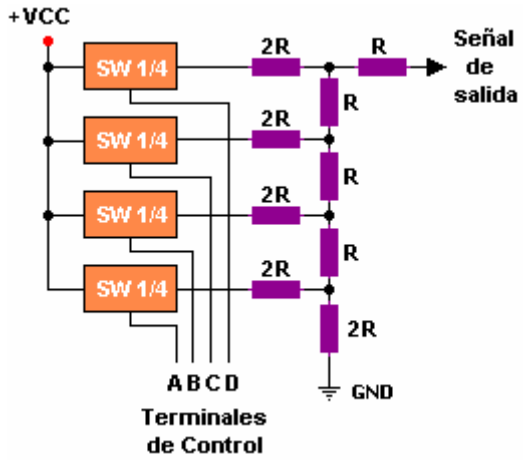
En este caso puedes utilizar señales digitales en los pines de control para seleccionar una de las cuatro señales analógicas presentes en los canales A, B, C o D y enviarlo como señal de salida, observa el siguiente esquema...



Conversor Digital/Analógico:

Otra de las aplicaciones, es implementarlo como un conversor D/A (Digital/Analógico) de 4 bit's, en donde las distintas combinaciones de los terminales de control entregan en la salida del circuito un nivel de tensión que corresponde particularmente a cada combinación.

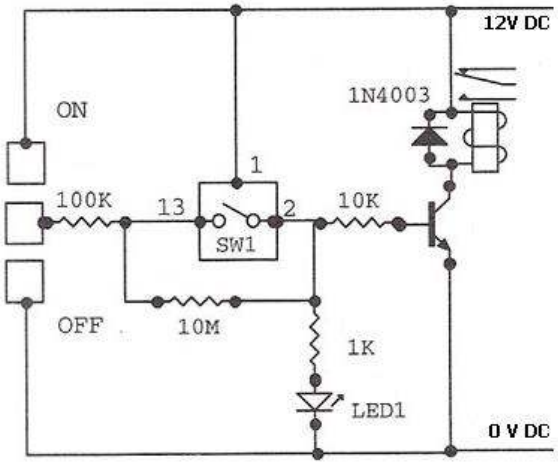
Veamos como sería la forma de conexión...



Pensándolo bien, creo que hasta puedes utilizarlo como un potenciómetro pero controlado digitalmente.

Algo más sofisticado con este circuito integrado y un puñado de componentes para la decoración del hogar.

Está de moda ocultar y sustituir los interruptores voluminosos. Con este integrado se añade una novedad y realza la belleza en la decoración de interiores. Hasta cuatro switches pueden ser configurados desde un solo circuito integrado. Los sensores de contacto podría ser sólo pequeños puntos de la realización de placas, por ejemplo de cobre, a sólo tres puntos por cada sensor. Los contactos del relé se pueden utilizar en lugar de los contactos del regulador del ventilador para hacer un control táctil de velocidad del ventilador o encender y apagar luces.



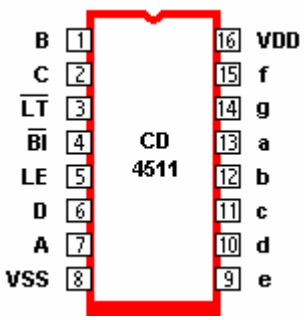
Bien, se pueden hacer más aplicaciones, por ejemplo un conmutador se señales de entrada, etc. Probad y me comentan el resultado, les estaré agradecidos, ahí queda...

Circuito Integrado CD4511

Presentación

Se trata de un excitador/decodificador/cerrojo BCD a 7 Segmentos.

El circuito provee las funciones de un cerrojo (latch) de almacenamiento de 4 bit, un decodificador BCD 8421 a 7 segmentos, y gran capacidad de excitación de salida.



Las entradas de prueba (LT), borrado (BI) y habilitación de Cerrojo (LE), se usan para probar el visualizador, para apagar o modular por pulsos el visualiador, y para almacenar un código BCD, respectivamente. Se puede usar con indicadores o Diodos LED

Entradas							Salidas							
LE	BI	LT	D	C	B	A	a	b	c	d	e	f	g	Visualiza
X	X	0	X	X	X	X	1	1	1	1	1	1	1	8
X	0	1	X	X	X	X	0	0	0	0	0	0	0	
0	1	1	0	0	0	0	1	1	1	1	1	1	0	0
0	1	1	0	0	0	1	0	1	1	0	0	0	0	1
0	1	1	0	0	1	0	1	1	0	1	1	0	1	2
0	1	1	0	0	1	1	1	1	1	1	0	0	1	3
0	1	1	0	1	0	0	0	1	1	0	0	1	1	4
0	1	1	0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	1	0	0	0	1	1	1	1	1	6
0	1	1	0	1	1	1	1	1	1	0	0	0	0	7
0	1	1	1	0	0	0	1	1	1	1	1	1	1	8
0	1	1	1	0	0	1	1	1	1	0	0	1	1	9
0	1	1	1	0	1	0	0	0	0	0	0	0	0	
0	1	1	1	0	1	1	0	0	0	0	0	0	0	
0	1	1	1	1	0	0	0	0	0	0	0	0	0	
0	1	1	1	1	0	1	0	0	0	0	0	0	0	
0	1	1	1	1	1	0	0	0	0	0	0	0	0	
0	1	1	1	1	1	1	0	0	0	0	0	0	0	
1	1	1	X	X	X	X	*							

Nota:

X = Sin Importancia

* = Depende del código BCD aplicado durante la transición de 0 a 1 de LE

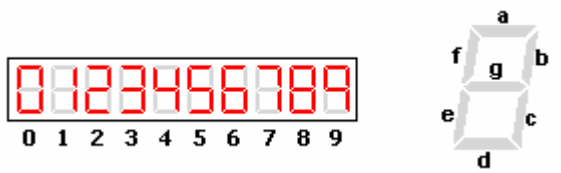
La segunda línea borra la salida, sin importar lo que hay en las entradas, el display se mantendrá apagado, esto es por culpa de BI=0 y LT=1.

Aquello que ves en azul, es el código para mostrar los respectivos valores en el display, los 6 siguientes son ignoradodos.

El último valor, depende del valor en que se encontraban las entradas en el momento de pasar

de 0 a 1 en el pin LE (Latch Enable).

Aquí tienes una imagen de lo que verías en el Display para cada valor de salida del integrado.

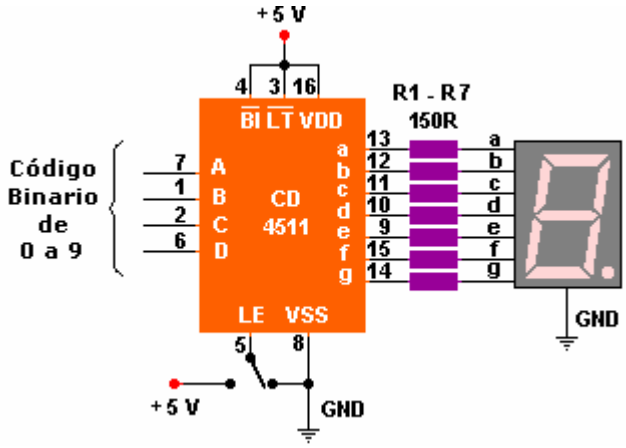


También la disposición de los segmentos y su identificación, nota que aquí no figura el punto en el segmento, será que quien lo fabricó se olvidó del puntito...???

Estas son otras características con las que cuenta este integrado:

- Baja disipación de potencia del circuito lógico
- Salidas que suministran alta corriente (hasta 25 mA)
- Almacenamiento de código en cerrojo
- Entrada de borrado
- Borrado del indicador en todas las condiciones ilegales de entrada
- Facilidad de tiempo compartido (multiplexado)
- Equivalente al Motorola MC14511

Para hacer pruebas con este integrado, puedes guiarte con el siguiente circuito...



Esto en caso de que el Display sea de Cátodo Común, de lo contrario necesitarás inversores, puesto que el código será totalmente distino, para ello, simplemente invirtiendo el valor de la salida, solucionas el problema.

En este circuito también tienes la posibilidad de probar que es lo que ocurre con el pin LE, si lo dejas como está en la imagen, trabajará normalmente, si lo cambias, se habilitará el cerrojo y retendrá el último valor ingresado, pero, deberías probarlo, y así comprenderás que es eso del cerrojo, ya que no es el único integrado que lo tiene, hay muchos más.

Creo que son datos suficientes para comprender como puedes implementarlos en tus proyectos.

Microcontroladores

Conceptos básicos

Llegó el gran momento...!!!. Antes de dar inicio a esta serie de capítulos que pretenden ser un "buen tutorial sobre microcontroladores", quiero agradecer a quienes colaboraron en la publicación de estas páginas y a quienes las enriquecerán con sus valiosos aportes, los cuales serán muy bienvenidos, así es que ... GRACIAS...!!!

Bien, Es hora de enfrentarse a la realidad... La idea es poder disponer de un tutorial completo de estos bichos, bueno, al menos de algunos de ellos, yo me centraré en uno en especial, y arrancaré de cero...

Armar circuitos electrónicos, muchas veces resulta interesante. El asunto es que... se pierde ese interés cuando los circuitos que pretendes armar resultan complejos, y entonces debes buscar nuevos componentes y te pierdes en la maraña de circuitos integrados (de ahora en más **IC**) que hay en el mercado.

Hasta que AL FIN...!!!, un día alguien te da a conocer una computadora metida en uno de ellos, y te dices ...tendré que conocerlos...!!!, y te das cuenta que tu vida cambió o cambiará, y nace una nueva pregunta... Deberé aprender programación...?, la respuesta ya la conoces... **OBVIO...!!!**

...Y así es como llegaste a esta página.

...que demás está decir que la iremos enriqueciendo entre todos ya que yo, sólo soy un aficionado en el tema...!!! (*modestia aparte*) :o)

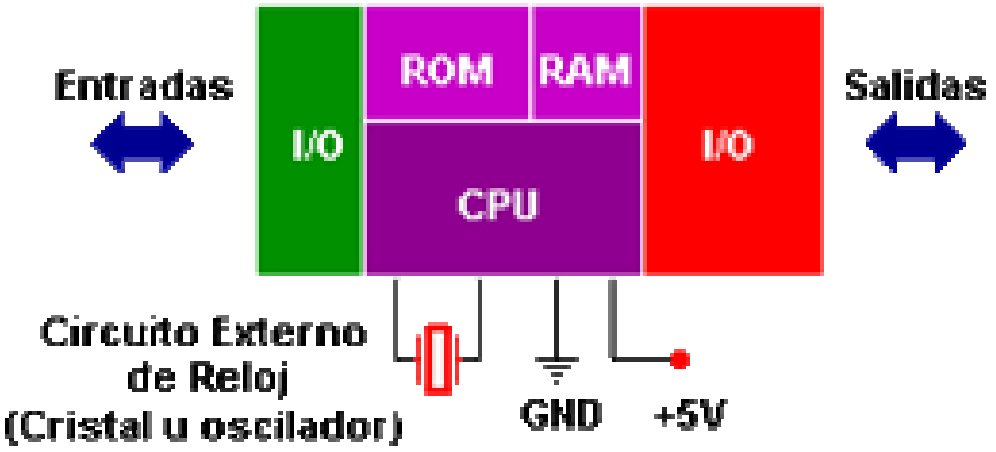
Sistemas microcontroladores

El diagrama de un sistema microcontrolador sería algo así

Los dispositivos de entrada pueden ser un teclado, un interruptor, un sensor, etc.

Los dispositivos de salida pueden ser LED's, pequeños parlantes, zumbadores, interruptores de potencia (tiristores, optoacopladores), u otros dispositivos como relés, luces, un secador de pelo, en fin.. lo que quieras.

Aquí tienes una representación en bloques del microcontroador, para que te des una idea, y puedes ver que lo adaptamos tal y cual es un ordenador, con su fuente de alimentación, un circuito de reloj y el chip microcontrolador, el cual dispone de su CPU, sus memorias, y por supuesto, sus puertos de comunicación listos para conectarse al mundo exterior.



Definamos entonces al microcontrolador; Es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Sacado de un libro...!!!. En fin estas son básicamente algunas de sus partes...

- **Memoria ROM** (Memoria de sólo lectura)
- **Memoria RAM** (Memoria de acceso aleatorio)
- **Líneas de entrada/salida (I/O)** También llamados puertos
- **Lógica de control** Coordina la interacción entre los demás bloques

Eso no es todo, algunos traen funciones especiales, ya hablaremos de ellas.

Microcontroladores PIC16CXX/FXX de Microchip

Me referiré a estos porque serán los que utilizaré aquí, (al menos por ahora). Estos micros pertenecen a la gama media y disponen de un set de 35 instrucciones, por eso lo llaman de tipo RISC (Reduced Instruction Set Computer) en entendible sería "Computador con Set de Instrucciones Reducido" pocas instrucciones pero muy poderosas, otras son de tipo CISC (Complex Instruction Set Computer - Computador con Set de Instrucciones Complejo), demasiadas instrucciones, y lo peor, difíciles de recordar.

Esta familia de microcontroladores se divide en tres rangos según la capacidad de los microcontroladores. El más bajo lo compone la familia 16C5X. El rango medio lo componen las familias 16C6X/ 7X/ 8X, algunos con conversores A/D, comparadores, interrupciones, etc. La familia de rango superior lo componen los 17CXX.

Estas son las funciones especiales de las cuales disponen algunos micros...

- **Conversores análogo a digital (A/D)** en caso de que se requiera medir señales analógicas, por ejemplo temperatura, voltaje, luminosidad, etc.
- **Temporizadores programables (Timer's)** Si se requiere medir períodos de tiempo entre eventos, generar temporizaciones o salidas con frecuencia específica, etc.
- **Interfaz serial RS-232.** Cuando se necesita establecer comunicación con otro microcontrolador o con un computador.
- **Memoria EEPROM** Para desarrollar una aplicación donde los datos no se alteren a pesar de quitar la alimentación, que es un tipo de memoria ROM que se puede programar o borrar eléctricamente sin necesidad de circuitos especiales.

- **salidas PWM (modulación por ancho de pulso)** Para quienes requieren el control de motores DC o cargas resistivas, existen microcontroladores que pueden ofrecer varias de ellas.
- **Técnica llamada de "Interrupciones"**, (ésta me gustó) Cuando una señal externa activa una línea de interrupción, el microcontrolador deja de lado la tarea que está ejecutando, atiende dicha interrupción, y luego continúa con lo que estaba haciendo.

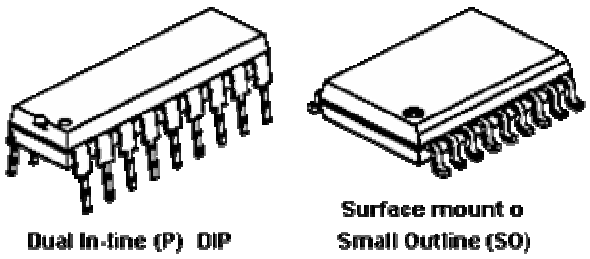
Todo esto, sólo para tener una idea de lo que son los micros, ahora veamos un par de ellos en especial

PIC16C84/F84

Aunque hoy en día existen en el mercado muchos micros, ninguno alcanzará la popularidad como el 16F84, este micro se utiliza en una gran variedad de circuitos y es por eso que es el elegido para presentarlo en este manual.

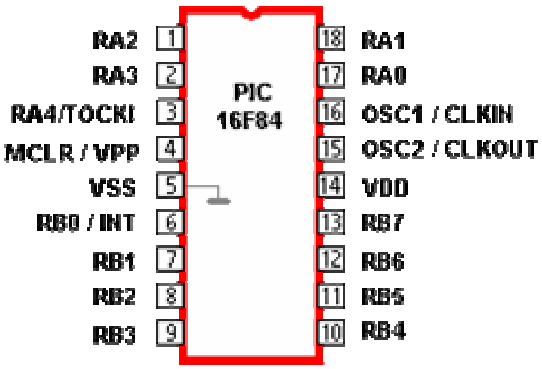
El PIC16C84 está fabricado en tecnología CMOS, consume baja potencia, y es completamente estático (si el reloj se detiene, los datos de la memoria no se pierden). El 16F84 tiene las mismas características pero posee memoria FLASH, esto hace que tenga menor consumo de energía, y como si fuera poco tiene mayor capacidad de almacenamiento.

El encapsulado más común para estos microcontrolador es el DIP (Dual In line Pin) de 18 pines, (el nuestro...), y utiliza un reloj de 4 MHz (cristal de cuarzo). Sin embargo, hay otros tipos de encapsulado, por ejemplo, el encapsulado tipo surface mount (montaje superficial) es mucho + pequeño.



Terminales del microcontrolador y sus respectivas funciones:

Ésta sería la disposición de sus terminales y sus respectivos nombres...



Encapsulado DIP - PIC16C84/F84

Patas 1, 2, 3, 17 y 18 (RA0-RA4/TOCKI): Es el PORT A. Corresponden a 5 líneas bidireccionales de E/S (definidas por programación). Es capaz de entregar niveles TTL cuando la alimentación aplicada en VDD es de 5V ± 5%. El pin **RA4/TOCKI** como entrada puede programarse en funcionamiento normal o como entrada del contador/temporizador TMR0. Cuando este pin se programa como entrada digital, funciona como un disparador de Schmitt (Schmitt trigger), puede reconocer señales un poco distorsionadas y llevarlas a niveles lógicos (cero y cinco voltios). Cuando se usa como salida digital se comporta como colector abierto; por lo tanto se debe poner una resistencia de pull-Up (resistencia externa conectada a un nivel de cinco voltios, ...no te preocupes, mas abajo lo entenderás mejor). Como salida, la lógica es inversa: un "0" escrito al pin del puerto entrega a la salida un "1" lógico. Este pin como salida no puede manejar cargas como fuente, sólo en el modo sumidero.

Pata 4 (MCLR / Vpp): Es una pata de múltiples aplicaciones, es la entrada de Reset (master clear) si está a nivel bajo y también es la habilitación de la tensión de programación cuando se está programando el dispositivo. Cuando su tensión es la de VDD el PIC funciona normalmente.

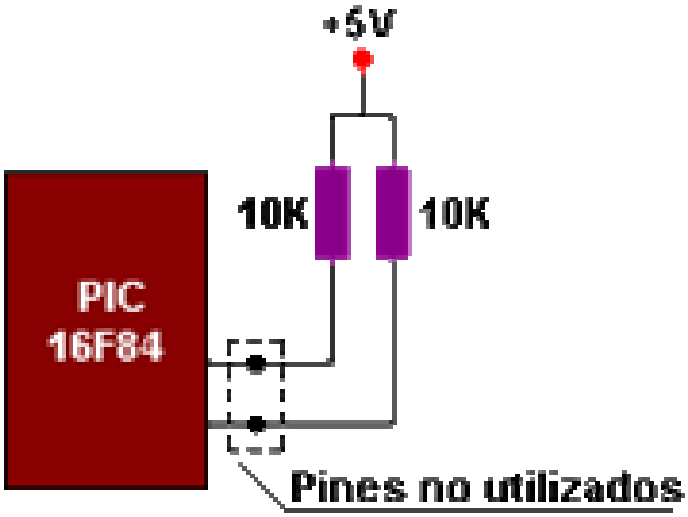
Patas 5 y 14 (VSS y VDD): Son respectivamente las patas de masa y alimentación. La tensión de alimentación de un PIC está comprendida entre 2V y 6V aunque se recomienda no sobrepasar los 5.5V.

Patas 6, 7, 8, 9, 10, 11, 12, 13 (RB0-RB7): Es el PORT B. Corresponden a ocho líneas bidireccionales de E/S (definidas por programación). Pueden manejar niveles TTL cuando la tensión de alimentación aplicada en VDD es de 5V ± 5%. RB0 puede programarse además como entrada de interrupciones externas INT. Los pines RB4 a RB7 pueden programarse para responder a interrupciones por cambio de estado. Las patas RB6 y RB7 se corresponden con las líneas de entrada de reloj y entrada de datos respectivamente, cuando está en modo programación del integrado.

Patas 15 y 16 (OSC1/CLKIN y OSC2/CLKOUT): Corresponden a los pines de la entrada externa de reloj y salida de oscilador a cristal respectivamente.

Un poco de electrónica:

Esto comienza a ponerse interesante, no crees...?, ok sigamos... Como estos dispositivos son de tecnología CMOS, todos los pines deben estar conectados a alguna parte, nunca dejarlos al aire porque se puede dañar el integrado. Los pines que no se estén usando se deben conectar a la fuente de alimentación de +5V, como se muestra en la siguiente figura...

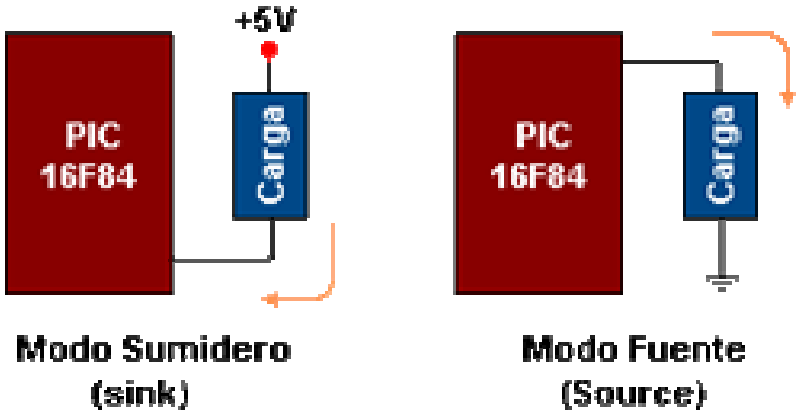


Capacidad de corriente en los puertos

La máxima capacidad de corriente de cada uno de los pines de los puertos en modo sumidero (sink) es de 25 mA y en modo fuente (source) es de 20 mA. La máxima capacidad de corriente total de los puertos es:

	PUERTO A	PUERTO B
Modo Sumidero	80 mA	150 mA
Modo Fuente	50 mA	100 mA

Así se vería la conexión para ambos modos de funcionamiento.



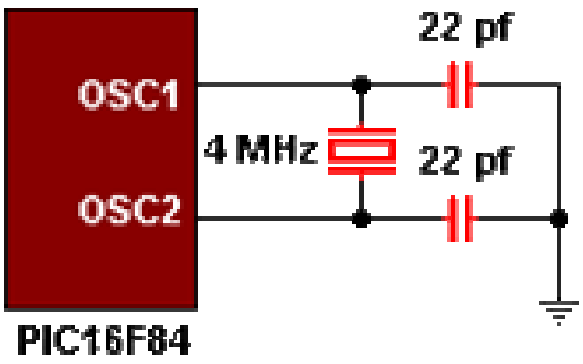
El oscilador externo

Es un circuito externo que le indica al micro la velocidad a la que debe trabajar. Este circuito, que se conoce como oscilador o reloj, es muy simple pero de vital importancia para el buen funcionamiento del sistema. El P1C16C84/F84 puede utilizar cuatro tipos de reloj diferentes. Estos tipos son:

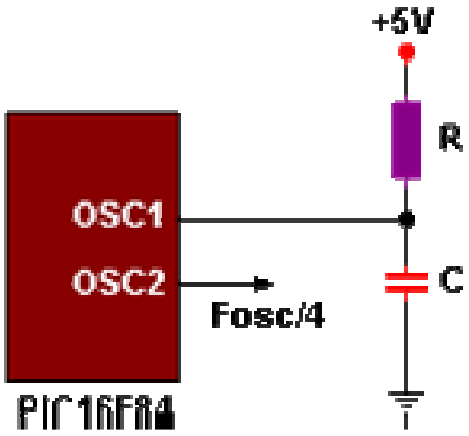
- **RC.** Oscilador con resistencia y condensador.
- **XT.** Cristal.
- **HS.** Cristal de alta velocidad.
- **LP.** Cristal para baja frecuencia y bajo consumo de potencia.

En el momento de programar o "quemar" el microcontrolador se debe especificar que tipo de oscilador se usa. Esto se hace a través de unos fusibles llamados "fusibles de configuración" o **fuses.**

Aquí utilizaremos el cristal de 4 MHz, porque garantiza mayor precisión y un buen arranque del microcontrolador. Internamente esta frecuencia es dividida por cuatro, lo que hace que la frecuencia efectiva de trabajo sea de 1 MHz, por lo que cada instrucción se ejecuta en un microsegundo. El cristal debe ir acompañado de dos condensadores y el modo de conexión es el siguiente.



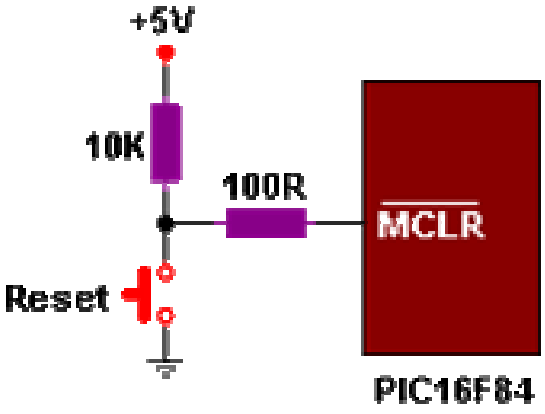
Si no requieres mucha precisión en el oscilador, puedes utilizar una resistencia y un condensador, como se muestra en la figura. donde OSC2 queda libre entregando una señal cuya frecuencia es la del OSC/4.



Según las recomendaciones de Microchip R puede tomar valores entre 5k y 100k, y C superior a 20pf.

Reset

El PIC 16C84/F84 posee internamente un circuito temporizador conectado al pin de reset que funciona cuando se da alimentación al micro, se puede entonces conectar el pin de MCLR a la fuente de alimentación. Esto hace que al encender el sistema el microcontrolador quede en estado de reset por un tiempo mientras se estabilizan todas las señales del circuito (lo cual es bastante bueno, por eso siempre la usaremos...).



Este último circuito, es por si deseas tener control sobre el reset del sistema, sólo le conectas un botón (pulsador) y listo...

Ahora vamos al interior del micro

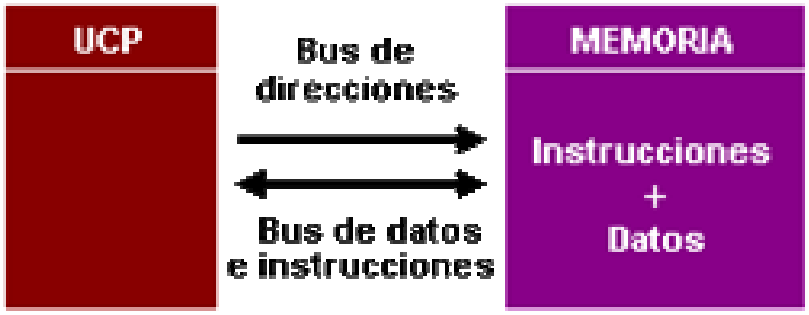
Estructura interna del Microcontrolador

Uffff...!!!, Ya se...!!!, tranquilo que ya comenzaremos con lo que estas esperando, antes debemos saber donde alojar nuestro programa, como se va a ejecutar, y como configurar sus puertos.

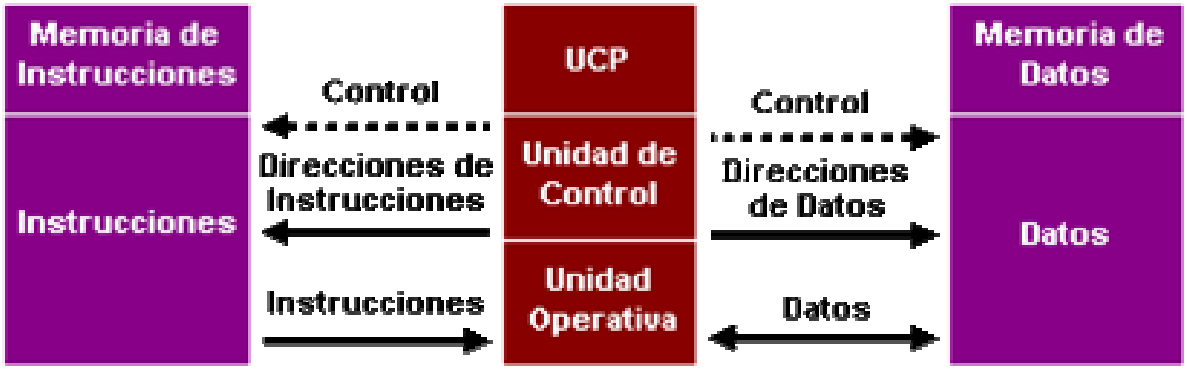
Arquitectura interna del PIC:

Hay dos arquitecturas conocidas; la clásica de von Neumann, y la arquitectura Harvard, veamos como son...

Arquitectura Von Neumann Dispone de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control).



Arquitectura Harvard Dispone de dos memorias independientes, una que contiene sólo instrucciones, y otra que contiene sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias, ésta es la estructura para los PIC's.



Ahora vamos por partes.

El procesador o UCP (CPU en inglés)

Es el elemento más importante del microcontrolador. Se encarga de direccionar la memoria de instrucciones, recibir el código OP de la instrucción en curso, decodificarlo y ejecutarlo, también realiza la búsqueda de los operandos y almacena el resultado.

Memoria de programa

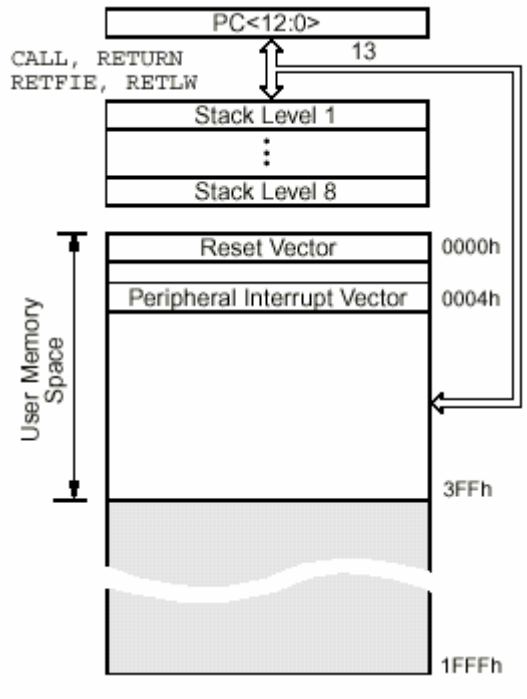
Esta vendría a ser la memoria de instrucciones, aquí es donde almacenaremos nuestro programa o código que el micro debe ejecutar. No hay posibilidad de utilizar memorias externas de ampliación. Son 5 los tipos de memoria. pero sólo describiré dos:

- **Memorias EEPROM.** (Electrical Erasable Programmable Read Only Memory - Memoria de sólo lectura Programable y borrable eléctricamente) Común en el PIC 16C84. Ésta tarea se hace a través de un circuito grabador y bajo el control de un PC. El número de veces que

puede grabarse y borrarse una memoria EEPROM es finito aproximadamente 1000 veces, *no es acaso suficiente...?*. Este tipo de memoria es relativamente lenta.

- **Memorias FLASH.** Disponible en el PIC16F84. Posee las mismas características que la EEPROM, pero ésta tiene menor consumo de energía y mayor capacidad de almacenamiento, por ello está sustituyendo a la memoria EEPROM.

La memoria de programa se divide en páginas de 2,048 posiciones. El PIC16F84A sólo tiene implementadas 1K posiciones es decir de 0000h a 03FFh y el resto no está implementado. (*es aquello que se ve en gris*)



Cuando ocurre un Reset, el contador de programa (PC) apunta a la dirección 0000h, y el micro se inicia nuevamente. Por esta razón , en la primera dirección del programa se debe escribir todo lo relacionado con la iniciación del mismo (*por ejemplo, la configuración de los puertos...*).

Ahora, si ocurre una interrupción el contador de programa (PC) apunta a la dirección 0004h, entonces ahí escribiremos la programación necesaria para atender dicha interrupción.

Algo que se debe tener en cuenta es la pila o Stack, que consta de 8 posiciones (*o niveles*), esto es como una pila de 8 platos el último en poner es el primero en sacar, si seguimos con este ejemplo, cada plato contiene la dirección y los datos de la instrucción que se está ejecutando, así cuando se efectúa una llamada (CALL) o una interrupción, el PC sabe donde debe regresar (*mediante la instrucción RETURN, RETLW o RETFIE, según el caso*) para continuar con la ejecución del programa.

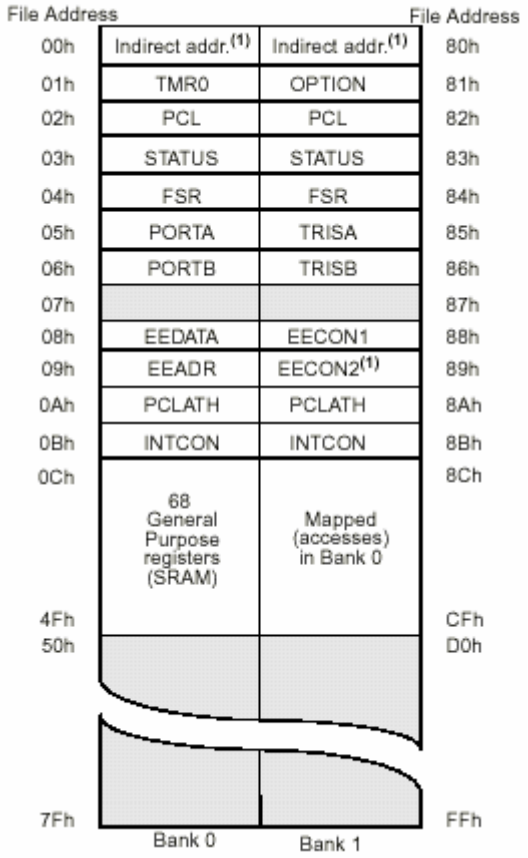
Recuerda, sólo 8 llamadas "CALL", ten en cuenta las "INTERRUPCIONES".

Memoria de datos

Tiene dos zonas diferentes:

1. RAM estática ó SRAM: donde residen los Registros Específicos (SFR) con 24 posiciones de tamaño byte, aunque dos de ellas no son operativas y los Registros de Propósito General (GPR)

con 68 posiciones. La RAM del PIC16F84A se halla dividida en dos bancos (banco 0 y banco 1) de 128 bytes cada uno (7Fh)



2. EEPROM: de 64 bytes donde, opcionalmente, se pueden almacenar datos que no se pierden al desconectar la alimentación.

Programación

Configuración de los puertos del PIC

Llegó el momento de ver como configurar los puertos del PIC. Para poder hacerlo es necesario conocer la tabla de registros de la memoria de datos, la cual como dijimos, está dividida en el **BANCO 0** y **BANCO 1**.

Los registros importantes en la configuración de los puertos son:

STATUS dirección **0x3**
PORTA dirección **0x5**
PORTB dirección **0x6**
TRISA dirección **0x5**
TRISB dirección **0x6**

Por defecto el PIC tendrá todos los I/O port's (es decir los puertos RA y RB), colocados como entrada de datos, y si queremos cambiarlos habrá que configurarlos.

Al configurar los puertos deberás tener en cuenta que:

Si asignas un **CERO (0)** a un pin, éste quedará como **salida** y...
Si le asignas un **UNO (1)**, quedará como **entrada**

Esta asignación se hace en:

TRISA para los pines del **PUERTO A** (5 bits)
TRISB para los pines del **PUERTO B** (8 bits)

Por Ejemplo:

Si **TRISA** es igual a 11110 todos sus pines serán entradas salvo RA0 que esta como salida

Si **TRISB** es igual a 00000001 todos sus pines serán salidas salvo RB0 que esta como entrada

Cuando el PIC arranca se encuentra en el BANCO 0, como TRISA y TRISB están en el **BANCO 1** no queda otra, deberemos cambiar de banco. Esto se logra a través del Registro **STATUS**

El registro STATUS

STATUS es un Registro de 8 bits u 8 casillas, en el cual la Nº 5 (**RP0**) define la posición del banco en donde nos encontramos

Si pones un **CERO (0)** a RP0 estaremos en el **BANCO 0**

Si le pones un **UNO (1)** ya ves, estaremos en el **BANCO 1**

REGISTRO STATUS							
7	6	5	4	3	2	1	0
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**

Listo, ahora ya sabemos como configurar los puertos, pero lo aclararemos con un ejemplo completo.

Código para configurar los puertos

Vamos a escribir un código que configure todos los pines del puerto A como entrada y todos los del puerto B como salida.

```
;--------------Encabezado-------------

        list    p=16f84     ; usaremos el PIC 16f84
        radix   hex         ; y la numeración hexadecimal

;------------mapa de memoria---------

estado  equ     0x03        ; Aquí le asignamos nombres a los
trisa   equ     0x05        ; registros indicando la posición
```

```
trisb       equ       0x06          ; en la que se encuentran

;-------Configuración de puertos-------

reset       org       0x00          ; origen del programa, aquí comenzaré
                                    ; siempre que ocurra un reset
            goto      inicio        ; salto a "inicio"
            org       0x05          ; origen del código de programa
inicio      bsf       estado,5                ; pongo rp0 a 1 y paso al
banco1
            movlw     b'11111'      ; cargo W con 11111
            movwf     trisa         ; y paso el valor a trisa
            movlw     b'00000000'    ; cargo W con 00000000
            movwf     trisb         ; y paso el valor a trisb
            bcf       estado,5                ; pongo rp0 a 0 y regreso al
banco0

;----------------------------------------
            end                     ; se acabó
;----------------------------------------
```

Descripción del código:

Todo lo que escribas luego de un **";"** (*punto y coma*) será ignorado por el ensamblador, estos son los famosos comentarios, y sirve para saber que hace cada línea de código.

Dicho esto no queda más que describir el código, así que vamos por partes.

```
;--------------Encabezado-------------

            list      p=16f84       ; usaremos el PIC 16f84
            radix     hex           ; y la numeración hexadecimal
```

Aquí le indicas al ensamblador para que microcontrolador estas codificando (PIC16F84). y cual será el sistema de numeración que utilizarás (hexadecimal).

Nota que hay tres columnas, en este caso la primera está vacía. Respeta las tabulaciones para no confundir al ensamblador.

```
;------------mapa de memoria---------

estado      equ       0x03          ; Aquí le asignamos nombres a los
trisa       equ       0x05          ; registros indicando la posición
trisb       equ       0x06          ; en la que se encuentran
```

Recuerdas lo de la memoria de datos...? Bien, al registro STATUS, que está en la posición 0x03 de la memoria de datos le puse la etiqueta **"estado"**. **equ** es algo así comoooo...**igual** . (Es decir, le estoy asignando el nombre estado al registro que está en la posición 0x03 de la memoria de datos).

Luego hice lo mismo con trisa y trisb. Ahora sigamos...

```
;-------Configuración de puertos-------

reset   org     0x00        ; origen del programa, aquí comenzaré
                            ; siempre que ocurra un reset
        goto    inicio      ; salto a "inicio"
        org     0x05        ; origen del código de programa
inicio  bsf     estado,5            ; pongo rp0 a 1 y paso al
banco1
        movlw   b'11111'    ; cargo W con 11111
        movwf   trisa       ; y paso el valor a trisa
        movlw   b'00000000'   ; cargo W con 00000000
        movwf   trisb       ; y paso el valor a trisb
        bcf     estado,5            ; pongo rp0 a 0 y regreso al
banco0
```

La directiva **org** indica el sitio de la memoria en donde se escribe una parte del programa. En este caso el contador de programa apuntará a la dirección 0x00 (*reset*) entonces ejecutará la instrucción que sigue a continuación, (saltar a la etiqueta inicio) y nuestro código de programa comienza en la dirección de memoria 0x05 (aquí salto por encima de la interrupción 0x04)

BSF (*SET FILE REGISTER*), es la instrucción que pone un uno en el bit del registro especificado, en este caso pone a uno el bit 5 del registro STATUS (*el rp0*), para pasar al banco 1.

movlw es algo así como... mueve el siguiente literal al Registro W.

W es el Registro de Trabajo, y lo usamos para almacenar momentáneamente los datos que queremos mover. una vez hecho esto pasamos el dato a trisa, o a trisb, según el caso.

movwf es algo así como... mueve el contenido del registro W al registro f, en este caso f sería trisa o trisb.

BCF (*BIT CLEAR FILE REGISTER*), ésta instrucción limpia el bit del registro especificado, o lo pone a cero, en este caso pone a cero el bit 5 del registro STATUS para regresar al banco 0.

```
;-----------------------------------------
        end                 ; se acabó
;-----------------------------------------
```

Programando en serio

Debo confesar que el programa anterior aunque parezca una burrada, lo utilizaremos de tiempo completo, y lo único que cambiaremos serán los pines de entrada y salida.

Te recuerdo que lo que hicimos hasta ahora, solo fue configurar los puertos, pero no genera ninguna señal ni nada por el estilo.

Ahora si programaremos en serio. Encenderemos un LED, lo mantendremos encendido por un tiempo, luego lo apagaremos y haremos que se repita todo de nuevo. Recuerda ponerle un nombre, aquí lo llamaré LED1.asm (no olvides el **.asm**)

Primer programa LED1.asm

```
;--------------Encabezado------------

            LIST      p=16f84
            radix     hex

;------------mapa de memoria---------

estado      equ       0x03          ; Haciendo asignaciones
TRISB       equ       0x06
ptob        equ       0x06

reg1        equ       0x0C          ; Estos 3 registros los utilizaré
reg2        equ       0x0D          ; para hacer el retardo
reg3        equ       0x0E

;-------Configuración de puertos-------

reset       org       0x00          ; origen del programa, aquí comenzará
                                    ; siempre que ocurra un reset
            goto      inicio        ; salta a "inicio"
inicio      bsf       estado,5              ; pone rp0 a 1 y pasa al
banco1
            movlw     b'00000000'   ; carga W con 00000000
            movwf     TRISB         ; y pasa el valor a trisb
            bcf       estado,5              ; pone rp0 a 0 y regresa al
banco0

;----Aquí enciende y apaga el LED-----

ahora       bsf       ptob,0        ; pone un 1 en RB0 (enciende el LED)
            call      retardo       ; llama al retardo

            bcf       ptob,0        ; pone a 0 RB0 (apaga el LED)
            call      retardo       ; llama al retardo
            goto      ahora         ; repite todo de nuevo

;----------Rutina de Retardo----------

retardo     movlw     10            ; Aquí se cargan los registros
            movwf     reg1          ; reg1, reg2 y reg3
                                    ; con los valores 10, 20 y 30
tres        movlw     20            ; respectivamente
            movwf     reg2

dos         movlw     30
            movwf     reg3

uno         decfsz    reg3,1        ; Aquí se comienza a decrementar
```

```
            goto    uno         ; Cuando reg3 llegue a 0
            decfsz  reg2,1      ; le quitare 1 a reg2
            goto    dos         ; cuando reg2 llegue a 0
            decfsz  reg1,1      ; le quitare 1 a reg1
            goto    tres        ; cuando reg1 llegue a 0
            retlw   00          ; regresare al lugar
                                ; de donde se hizo la llamada

;-----------------------------------------
            end                 ; se acabó
;-----------------------------------------
```

Descripción del código:

No te asustes por el tamaño del código, que aunque parezca difícil todo está igual que el código anterior, por lo que sólo describiré los cambios... (**lo que está en rojo**)

Se agregaron 3 registros mas (reg1, reg2 y reg3), éstos vendrían a ser como variables ubicadas en sus respectivas posiciones (0x0C, 0x0D, 0x0E,) y son registros de propósito general (recuerda que para el PIC16F84 son 68, puedes elegir cualquiera).

A demás se agregó **ptob**, etiqueta que corresponde a la dirección del puerto B

Analicemos lo que sigue..., que éste es el programa en sí:

```
;----Aquí enciende y apaga el LED-----

ahora   bsf     ptob,0      ; pone un 1 en RB0 (enciende el LED)
        call    retardo     ; llama al retardo

        bcf     ptob,0      ; pone a 0 RB0 (apaga el LED)
        call    retardo     ; llama al retardo
        goto    ahora       ; repite todo de nuevo
```

La etiqueta **"ahora"** es el nombre de todo este procedimiento o **rutina**, de tal modo que cuando quiera repetir el procedimiento solo saltare a **"ahora"**.

bsf es poner a uno un bit, en este caso al primer bit (el bit **0**) del puerto B (**ptob**).

call es una llamada, en este caso llama a la rutina de **retardo**, cuando regrese, continuará con el código.

bcf es poner a cero un bit, en este caso al primer bit (bit **0**) del puerto B (**ptob**). y luego llama al retardo, cuando regrese se encontrará con la instrucción **goto** obligándolo a saltar a la etiqueta **ahora** para que se repita todo de nuevo. Eso es todo...!!!.

Rutina de retardo

Esta es la parte más difícil, pero trataré de hacerlo sencillo así puedes continuar con lo que sigue y no te trabas en esta parte. **Primero veremos como se cargan los registros para el retardo.** Veamos el código...

```
;----------Rutina de Retardo----------

retardo   movlw   10          ; Aquí se cargan los registros
          movwf   reg1        ; reg1, reg2 y reg3
                              ; con los valores 10, 20 y 30
tres      movlw   20          ; respectivamente
          movwf   reg2

dos       movlw   30
          movwf   reg3
```

Recordemos que en el mapa de memoria los registros **0x0C, 0x0D** y **0x0E** fueron nombrados como **reg1, reg2** y **reg3** respectivamente. Ahora simularemos los tres registros para ver como se cargan mediante el registro de trabajo W, (utilizamos W por que los valores **10, 20** y **30** son valores constantes). Repito, esto es una simulación bien a lo bruto, así que vamos a suponer que en vez de 10 cargo **1**, en lugar de 20 cargo **2** y en lugar de 30 cargo **3**, hago esto, solo con fines didácticos así podrás comprenderlo mejor, ok?.

Lo que acabas de ver, fue la carga de los registros reg1, reg2 y reg3. Ahora verás como se comienza a decrementar cada uno de esos registros, primero reg3, luego reg2 y finalmente reg1.

```
tres      movlw   20          ; respectivamente
          movwf   reg2

dos       movlw   30
          movwf   reg3

uno       decfsz  reg3,1      ; Aquí se comienza a decrementar
          goto    uno         ; Cuando reg3 llegue a 0
          decfsz  reg2,1      ; le quitare 1 a reg2
          goto    dos         ; cuando reg2 llegue a 0
          decfsz  reg1,1      ; le quitare 1 a reg1
          goto    tres        ; cuando reg1 llegue a 0
          retlw   00          ; regresare al lugar
                              ; de donde se hizo la llamada
```

Veamos, **decfsz reg3,1** esto es, decrementa reg3, si al decrementar te da cero saltéate una línea. El **1** que sigue a reg3, indica que guarde el valor de reg3 decrementado en el mismo reg3, es comoooo.... contador=contador-1 (**se entiende...?**)

goto, es saltar y **goto uno** es saltar a la etiqueta **uno**. En esta pequeña vuelta estoy decrementando reg3 hasta que se haga cero.

Cuando reg3 llegue a 0 decrementaré reg2 en una unidad, volveré a cargar reg3 y lo decrementaré nuevamente para recién restarle otra unidad a reg2, y así... hasta que reg2 se haga cero. Cuando eso ocurra decrementaré reg1 en una unidad, cargaré nuevamente reg2 y reg3, para luego decrementarlos de nuevo, todo esto ocurrirá hasta que reg1 se haga igual a cero.

Luego de cargar los 3 registros el siguiente código es...

```
uno     decfsz  reg3,1
```

y así continua...

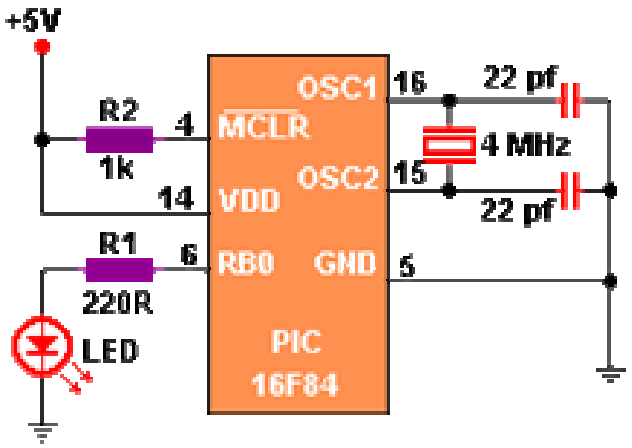
Esta rutina de retardo, aunque parezca absurda y larga nos permite ver como se enciende y se apaga el LED, de lo contrario no podríamos notar la diferencia, o lo veríamos apagado o encendido, ya que la velocidad es demasiado alta si estamos trabajando con un XT de 4 MHz. Finalmente nos queda la última instrucción:

```
;-----------------------------------------
            end                     ; se acabó
;-----------------------------------------
```

Sin palabras.

Esquema eléctrico para LED1

Una vez cargado el programa en el PIC, necesitarás ver el programa funcionando, por lo que deberás armar este circuito.



El pin 4 (MCLR) está conectado por lo del Reset, para que se estabilicen los niveles de tensión.

Herramientas de programación

Este apartado de herramientas para compilar el código asm y pasarlo a hexadecimal es para trabajar en modo MS-DOS, lo dejo como una pequeña orientación a la programación del PIC, estas herramientas aún se encuentran en la RED y son muy eficaces y sencillas. Existe mucho software en el mercado y estupendos manuales para programar los PIC, uno de ellos es el de **PIC's para principiantes** que tambien está disponible en **www.mikroe.es** para descargarlo gratis. Reconozco que hoy día muchos usuarios sólo están acostumbrados a usar el PC en modo windows, no hay problema, se saltan este apartado y buscan el libro que les he recomendado anteriormente.

Lo anterior estuvo bárbaro, pero dónde voy a escribir el código?, como se hace para ensamblar?, y cómo cargo mi programa en el PIC?, mmmmm... demasiadas preguntas, comencemos de cero...

Necesitaras:

- Un PC (con una 386 más que suficiente).
- El programa para editar tu código, que bien podría ser el **Edit** del DOS y así generar tu archivo .asm
- Como ensamblador, yo utilizo es **Mpasm 2.15** y puedes bajarlo de www.microchip.com y con éste me basta para generar los archivos .hex
- El programa para grabar el PIC, Con el **Prog V.1.41** no tuve problemas y va perfecto con mi circuito grabador, puedes buscarlo en www.webelectronica.com.ar, ahora también disponible desde aquí
- Por supuesto necesitas el Hardware (circuito electrónico que se conecta al puerto de la PC) en el que insertarás el PIC para cargar el programa, hay muchísimos por la red, pero si quieres el que yo utilizo, **aquí** tienes el esquema eléctrico, el listado de componentes y la conexión al puerto paralelo, si ya lo quieres montado y listo para usar, contacta conmigo, que más puedo decir...

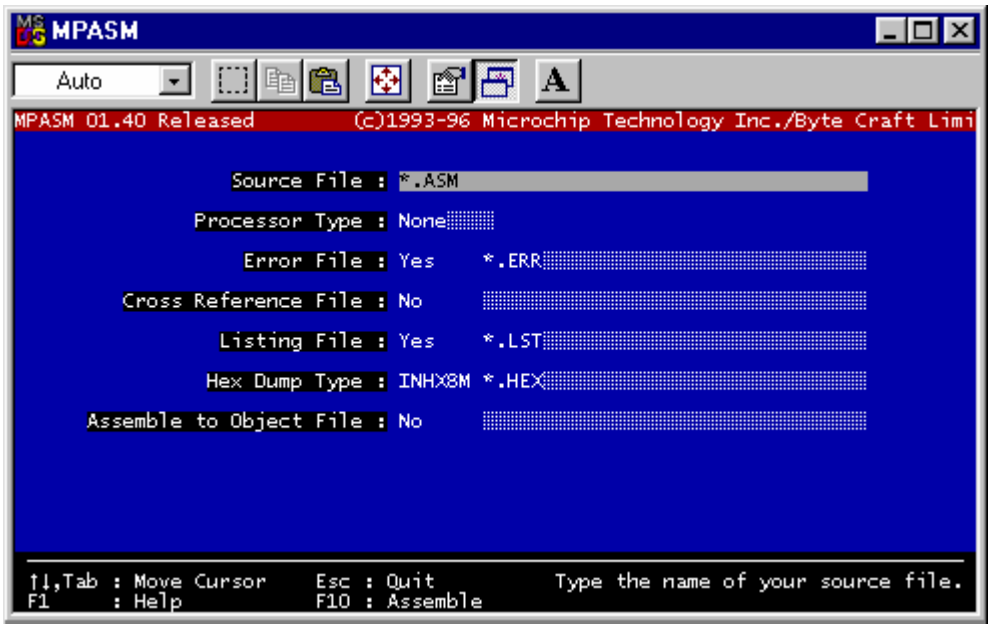
Abre una ventana del DOS y apunta al directorio donde tienes todas tus herramientas, yo las puse en una carpeta llamada **tutor**, si haces lo que yo, te quedará algo así.

C:\tutor>edit

Una vez ahí puedes escribir tu código, por último lo guar-damos seleccionando el menú **Archivo --> Guardar como --> led1.asm** No olvides el .asm

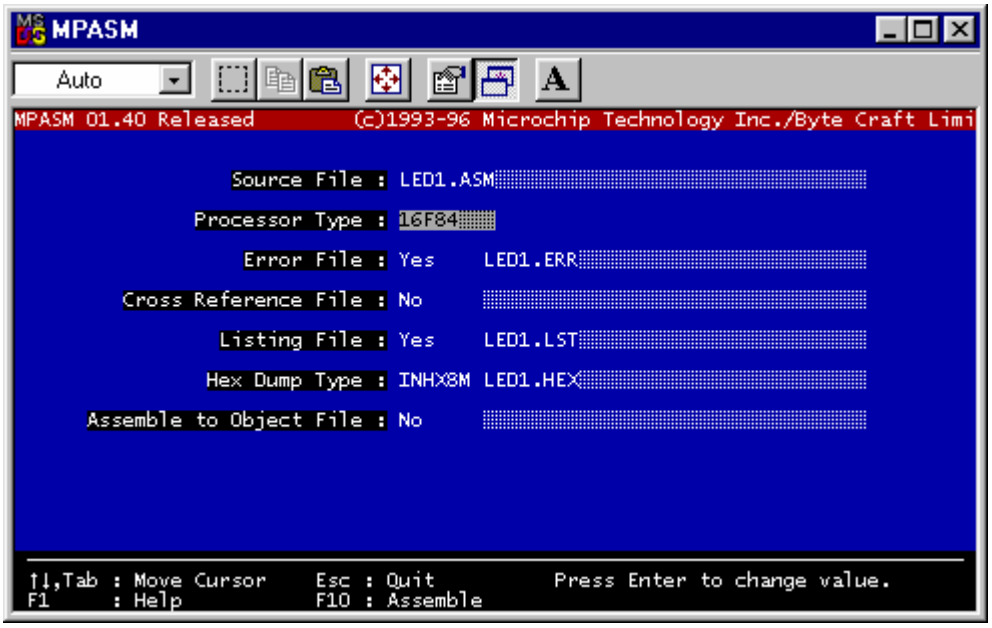
MS-DOS - EDIT
Auto
Archivo Edición Buscar Ver Opciones Ayuda
C:\tutor\led1.asm
;---------------Encabezado-------------
LIST P=16f84
;------------mapa de memoria---------
estado equ 0x03 ; Haciendo asignaciones
TRISB equ 0x06
ptob equ 0x06
reg1 equ 0x0C ; Estos 3 registros los utilizaré
reg2 equ 0x0D ; para hacer el retardo
reg3 equ 0x0E
;-------Configuración de puertos-------
reset org 0x00 ; origen del programa, aquí comenzaré
; siempre que ocurra un reset
goto inicio ; salta a "inicio"
inicio bsf estado,5 ; pone rp0 a 1 y pasa al banco1
movlw b'00000000' ; carga W con 00000000
movwf TRISB ; y pasa el valor a trisb
F1=Ayuda
Línea:2 Col:1

Muy bien, ya tenemos **led1.asm**, sólo resta ensamblarlo. Entonces vamos por Mpasm, lo abres, y veras algo como esto...


MPASM
Auto
MPASM 01.40 Released (c)1993-96 Microchip Technology Inc./Byte Craft Limi
Source File : *.ASM
Processor Type : None
Error File : Yes *.ERR
Cross Reference File : No
Listing File : Yes *.LST
Hex Dump Type : INHX8M *.HEX
Assemble to Object File : No
↑↓,Tab : Move Cursor
F1 : Help
Esc : Quit
F10 : Assemble
Type the name of your source file.

En **Sourse File** presiona Enter para seleccionar el archivo a ensamblar.

Haz lo mismo en **Processor Type** y busca el PIC16f84, que es el que usaremos, el resto lo dejas como está..., te debería quedar algo así...


MPASM
Auto
MPASM 01.40 Released (c)1993-96 Microchip Technology Inc./Byte Craft Limi
Source File : LED1.ASM
Processor Type : 16F84
Error File : Yes LED1.ERR
Cross Reference File : No
Listing File : Yes LED1.LST
Hex Dump Type : INHX8M LED1.HEX
Assemble to Object File : No
↑↓,Tab : Move Cursor
F1 : Help
Esc : Quit
F10 : Assemble
Press Enter to change value.

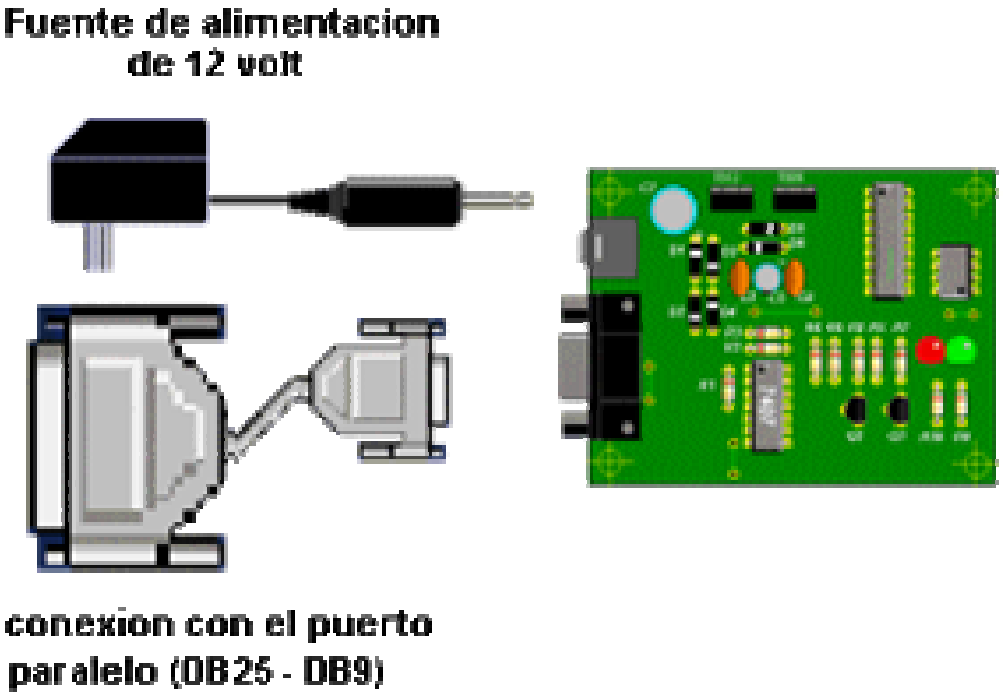
Ésto, generará el archivo **LED1.ERR, LED1.LST**, y **LED1.HEX**, este último es el que estamos buscando, los anteriores sirven para saber si se cometió algún error, si es así debes abrirlos (con el Bloc de Notas es suficiente) corregir los errores y ensamblar nuevamente.

Para ensamblar sólo debes presionar **F10** y verás la ventana de resultados

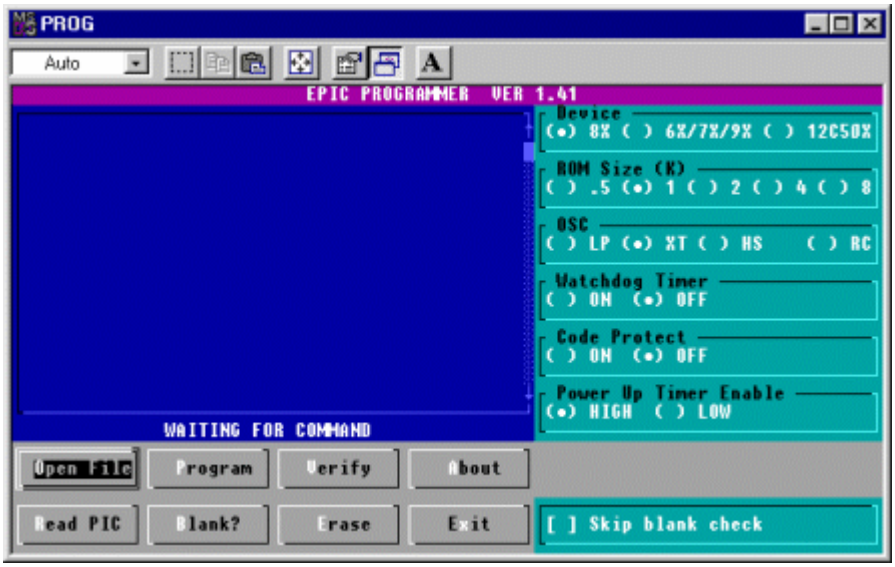
Que como es lógico... no cometí errores =P. Aunque por ahí apareció un Warning, que en realidad no es causa de falla en el código.

Cargando Led1.hex en el PIC

Bien, ya tenemos **LED1.HEX** y es el que cargaremos en el pic. Lo que viene es una imagen representativa del grabador de pic, con el cable que se conectará al puerto paralelo, y la fuente de alimentación externa. No coloques el PIC sin haber hecho las conexiones anteriores.



Ya puedes ejecutar el software de programación, abre **Prog.exe** y seguramente se apagarán los LED's rojo y verde (si estaban encendidos...!). Te aparecerá la siguiente ventana



haciendo click en **Open File** seleccionas **LED1.HEX** y aquí está...

Colocas el PIC en el grabador, luego Seleccionas la opción **Program** y esperas a que te aparezca el mensaje **Programming Complete**

Ejemmmmm, perdón... este mensajito salió porque el pic ya tenía un programa grabado, bueno, no importa, como es regrabable, sólo le daremos a ok y listo, el nuevo programa borrará al anterior.

Ahora siiiiii...!!!, Si todo fue bien, ya puedes quitar el PIC del zócalo y llevarlo a tu circuito para ver como funciona.

Los fusibles de progrmación

Algo que me estaba quedando en el tintero son los fusibles de programación, como:

Device el tipo de chip, en nuestro caso el 8x;
Rom Size la cantidad de memoria, en nuestro caso de 1k;
OSC el tipo de oscilador que utilizaremos, para nosotros un XT;
Watchdog Timer El perro guardián, que aquí no lo utilizamos;
Code protect para que nadie tenga acceso al código grabado en el PIC;
Power Up Timer temporizador de encendido. En el PIC16f84, funciona de modo invertido, por eso está en **LOW**. Para mayor detalle consulta la sección los fuses del PIC 16F84.

Ahora que ya estamos listos y preparados con todo el soft y el hard necesario, lo único que necesitamos es complicarnos un poco mas las cosas, allá vamos.

Los FUSES del PIC con mayor detalle

Estas 4 "variables" del Pic16F84 (modelos superiores tienen más), sirven para configurar ciertos aspectos del microcontrolador. Cada FUSE activa o desactiva una opción de funcionamiento.

OSC (Oscilador):

Es el modo de oscilación que va a usar el Pic.

Cada vez que el Pic recibe un pulso eléctrico del oscilador da un paso para ejecutar una instrucción (4 impulsos para completar una), por lo que podemos decir que es una señal que le recuerda al Pic que tiene que seguir avanzando.

Según esto, el pic puede usar 4 tipos de oscilador:

XT: Es un acrónimo que viene de XTAL (o cristal en castellano). Este modo de funcionamiento implica que tendremos que disponer de un cristal de cuarzo externo al

Pic y dos condensadores. El valor del cristal generalmente será de 4Mhz o 10Mhz, y los condensadores serán cerámicos de entre 27 y 33 nF. La exactitud de este dispositivo es muy muy alta, por lo que lo hace muy recomendable para casi todas las aplicaciones.

RC: Este es el sistema más sencillo y económico. Se basa en un montaje con una resistencia y un condensador. La velocidad a la que oscile el pic dependerá de los valores del condensador y de la resistencia. En la hoja de características del Pic están los valores.

HS: Para cuando necesitemos aplicaciones de "alta velocidad", entre 8 y 10Mhz. Se basa también en un cristal de cuarzo, como el XT

LP: "Low Power" la velocidad máxima a la que podemos poner el pic con este oscilador es de 200Khz. Al igual que el XT y el HS, necesitaremos de un cristal de cuarzo y unos condensadores.

WDT (Watchdog Timer):

El famoso "perro" del pic. (perro guardián). Esta es una capacidad del pic de autoresetearse. Es muy útil, por ejemplo si un Pic, por un descuido de programación, se queda en un bucle infinito, esta "utilidad" lo sacará de él. Su funcionamiento es sumamente sencillo. Simplemente es un registro que debemos borrar cada cierto tiempo. Si transcurrido un cierto tiempo el registro no ha sido borrado el pic se resetea. La instrucción para borrar el registro es CLRWDT. Con poner un par de ellos a lo largo de nuestro código es suficiente para tener una garantía de que el pic no se quede "haciendo el bobo" (como dijo alguien por ahí...).

PWRT (Power Up Timer Reset):

Si activamos este FUSE, lo que conseguimos es que se genere un retardo en la inicialización del Pic. Esto se usa para que la tensión se estabilice, por lo que se recomienda su uso.

CP (Code Protect):

El "dichoso" Code Protection. Protección del código. Lo único que hace es impedir que algún curioso, se apropie de tu creación no tiene efecto alguno en el correcto funcionamiento del PIC, ni que no se pueda sobrescribir su contenido. Lo único que hace es eso, impedir su lectura. Por cierto, dicen que puedes quitar la protección por medio de hardware, yo nunca lo hice, por que no lo creo necesario, ya que lo entretenido de esto es el desafío, no crees...???

Esperabas algo más...?

y bueno...!!!

Luego de enterarme que le llamaron **Obsoleto** a mi querido **16F84** que esperabas...!!!

Caí en depresión...!!!

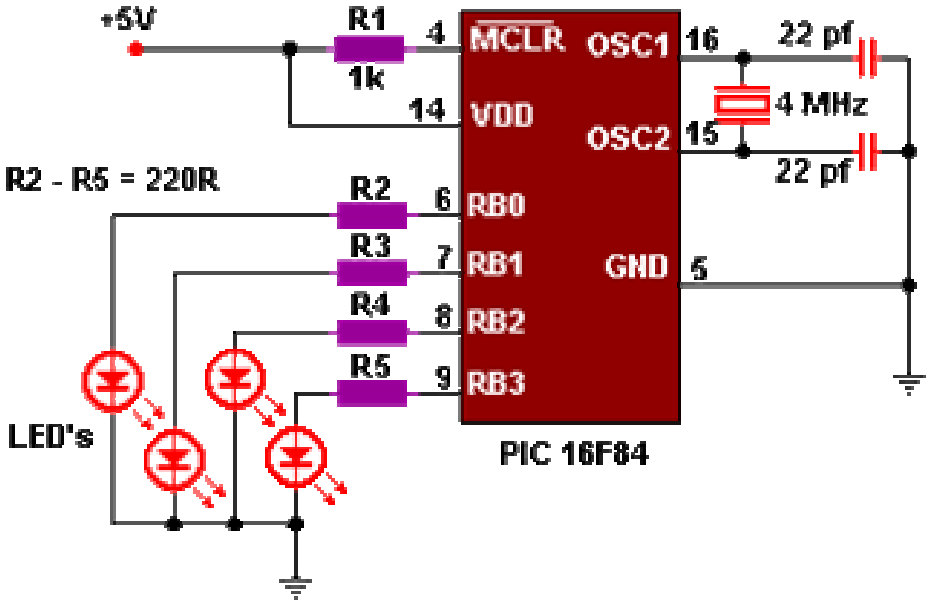
Segundo programa – LED4

Antes de empezar…

Si bien nos vamos a complicar con las siguientes lecciones, sería bueno que consultes el Set de instrucciones, así será mas fácil que comprendas cada línea de código que se escribe, ya que cada una está acompañada de su respectivo ejemplo, y una vez lo comprendas puedes quedarte con el Resumen de instrucciones.

Lo que haremos ahora será un programa que encienda 4 LED's en forma secuencial, y para ello recurriremos a la rutina de retardo del programa anterior, que espero lo hayas comprendido, si no es así regresa y no vengas aquí hasta que lo entiendas :o))

Ok, sigamos. El circuito será el siguiente...



Y el código que realiza la secuencia es el que viene a continuación.

```
;--------------Encabezado------------

            LIST      p=16f84
            radix     hex

;-----------mapa de memoria---------

estado      equ       0x03          ; Haciendo asignaciones
TRISB       equ       0x06
ptob        equ       0x06
rotar       equ       0x0A          ; para desplazar el dato

reg1        equ       0x0C          ; para hacer el retardo
reg2        equ       0x0D
reg3        equ       0x0E
```

```
;-------Configuración de puertos-------

reset   org     0x00

        goto    inicio          ; salta a "inicio"
        org     0x05            ; aquí comienza el programa
inicio  bsf     estado,5                ; configurando el puerto B
        movlw   b'00000000'
        movwf   TRISB
        bcf     estado,5

;----Realiza la secuencia de LED's-----

ahora   movlw   0x01            ; carga W con 00000001
        movwf   rotar           ; lo pasa al registro rotar

rotando movf    rotar,0         ; pasa el contenido de rotar a W
        movwf   ptob            ; y de allí al puerto B
                                ; encendiendo el LED correspondiente
        call    retardo         ; llama a retardo
        rlf     rotar,1         ; desplaza un bit el contenido
                                ; de rotar y lo guarda.
        btfss   rotar,4         ; prueba si se activa el 5to. bit
                                ; si es así saltea una línea
        goto    rotando         ; sino, sigue rotando
        goto    ahora           ; repite todo de nuevo

;-----------Rutina de Retardo----------

retardo movlw   10              ; Carga los registros
        movwf   reg1            ; reg1, reg2 y reg3
                                ; con los valores 10, 20 y 30
tres    movlw   20              ; respectivamente
        movwf   reg2

dos     movlw   30
        movwf   reg3

uno     decfsz  reg3,1          ; Comienza a decrementar
        goto    uno             ; cuando termine
        decfsz  reg2,1          ; regresará a la siguiente
        goto    dos             ; línea de código
        decfsz  reg1,1          ; de donde fue llamado
        goto    tres
        retlw   00

;----------------------------------------
        end                     ; The End
;----------------------------------------
```

Todo está como antes, salvo lo que está en rojo. Veamos de que se trata eso de rotar y rotando.

```
;----Realiza la secuencia de LED's-----

ahora    movlw   0x01        ; carga W con 00000001
         movwf   rotar       ; lo pasa al registro rotar

rotando  movf    rotar,0     ; pasa el contenido de rotar a W
         movwf   ptob        ; y de allí al puerto B
                             ; encendiendo el LED correspondiente
         call    retardo     ; llama a retardo
         rlf     rotar,1     ; desplaza un bit el contenido
                             ; de rotar y lo guarda.
         btfss   rotar,4     ; prueba si se activa el 5to. bit
                             ; si es así saltea una línea
         goto    rotando     ; sino, sigue rotando
         goto    ahora       ; repite todo de nuevo
```

La rotación

rotar es el registro en el que almacenaré momentáneamente el valor del desplazamiento de los bit's. Así cuando llegue al 5to. no enviaré ese dato ya que se habrá activado el 4to. bit del puerto B del PIC (sería el 4to. LED), y entonces volveré a comenzar.

Descripción del código:

Ok, voy a poner a 1 el primer bit del registro **rotar** a través de **w**, esto se hace en las dos primeras líneas.

rotando, es una subrrutina: Aquí se pasa el contenido del registro **rotar** o sea (**00000001**) a **W** (por eso el **0**) para luego enviarlo al puerto B (**portb**), y encender el primer LED, luego llama al retardo, cuando regrese se encontrará con la siguiente instrucción.

rlf rotar,1 esto es como decirle, rota el contenido del registro **rotar** un lugar a la izquierda, y guarda el valor en el mismo registro **rotar** (por eso el **1**). Es decir, que si antes **rotar=00000001** luego de esta instrucción **rotar=00000010**. Ahora viene la verificación del 5to. bit, para saber si completó la rotación.

btfss rotar,4 es como si le preguntarías ¿oye, se activó tu 5to. bit?.

Ok...!!! ya se lo que estas pensando ¿como que el 5to. si aparece el 4?, bueno, es por que no estas siguiendo el tutorial, recuerda que el primer bit está en la posición 0 y por ende, el 5to. esta en la posición 4 ¿ahora esta bien?. Continuemos, si resulta ser que no, saltara hasta **rotando** y pasará el contenido del registro rotar a W nuevamente (recuerda que ahora **rotar** es **00000010** por aquello del desplazamiento). luego lo enviará al puerto B, llamará al retardo y rotará nuevamente.

Bien supongamos que ya paso todo eso y que ahora **rotar** tiene el valor **00001000** y estamos ubicados justo en la etiqueta rotando. Entonces pasará el valor a **W** y de allí al puerto B, luego llamará al retardo, cuando regrese, rotará el contenido del registro **rotar** una vez más y entonces su contenido será **00010000**. Ahora viene la pregunta...

btfss rotar,4 ¿está activo el 5to. bit del registro **rotar**?, Siiiiii, si Sr. está activo..., Perfecto, entonces saltaré una línea, me encontraré con **goto ahora** y comenzaré de nuevo.

Eso es todo, ahora veámoslo en forma gráfica para aclarar un poco las ideas.

Codigo

```
ahora    movlw    0x01
         movwf    rotar

rotando  movf     rotar,0
         movwf    ptob
         call     retardo

         rlf      rotar,1

         btfss    rotar,4
         goto     rotando
         goto     ahora
```

Retardo

Registro 0x0A

rotar 00000000

Registro de Trabajo

w 00000000

Puerto B

RB7 ---------- RB0

ptob 00000000

Esa fue la idea, que veas como se hace la rotación, hay varias formas de hacerlo, yo aquí te mostré una, otras utilizan el **carry** del registro **STATUS**, otras no utilizan la rotación para hacer esta secuencia, sino van activando los bit's de a uno, en fin, tienes muchas opciones...

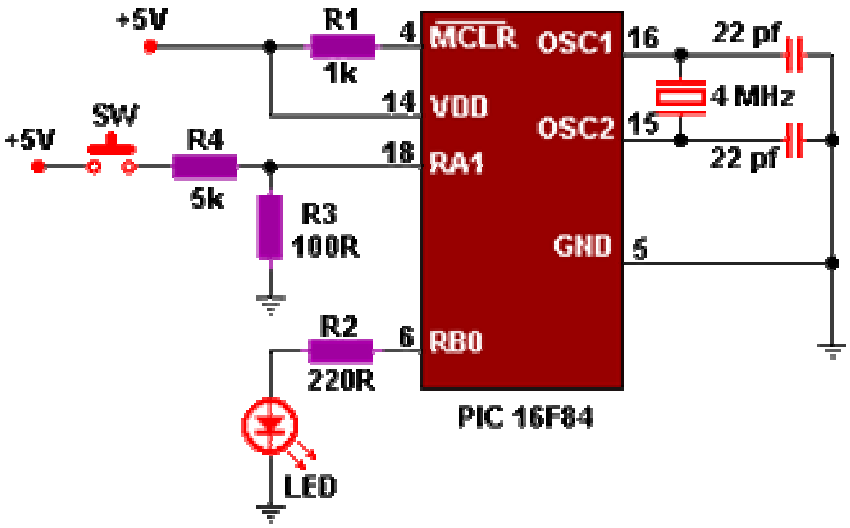
Señales de Entrada

Lo interesante y llamativo de los microcontroladores es que obedecen tus órdenes al pie de los bit's :o)

Por ejemplo, si le ordenas que vigile un pulsador, el muy pillo lo hará, y cuando alguien lo active le dará su merecido, jejejeje

Ahhhhhhh...!!!, y para complicarlo un poco más lo haremos con un solo pulsador. Si un travieso lo activa encenderemos un LED, y si lo activan nuevamente, lo apagaremos, quedó...? Mmmmmmmm... Lo que estoy notando es que voy a necesitar un registro que me guarde la información de si el LED está prendido o apagado, (sino... cómo voy a saber que hacer cuando lo presionen...!!!) bueno, ahora si...

Comencemos... el pulsador lo conectaré a RA1 y el LED a RB0, así se ve en el siguiente esquema



Hay varias formas de hacerlo, y más adelante utilizaremos el método de **Interrupciones.**

Programa para verificar el estado de un pulsador

```
;---------------Encabezado-------------
            List      p=16F84
            radix     hex
;------------mapa de memoria---------
STATUS EQU    03            ; status esta en la dirección 03
TRISA  EQU    05
PORTA  EQU    05            ; el puerto a
TRISB  EQU    06
PORTB  EQU    06            ; el puerto b

cont       EQU    0A

;-------Configuración de puertos-------

reset      ORG    0
           GOTO   inicio
           ORG    5

inicio     BSF    STATUS,5   ; configurando puertos
           MOVLW  0x02       ; carga w con 0000 0010
           MOVWF  TRISA      ; ahora RA1 es entrada
           MOVLW  0x00       ; carga w con 0000 0000
           MOVWF  TRISB      ; y el puerto B salida
           BCF    STATUS,5

           CLRF   PORTB      ; limpio el puerto B
           CLRF   cont       ; limpio el contador
           BSF    cont,0     ; pongo el contador a 1

;----------------------------------------

test       BTFSC  PORTA,1    ; si alguien presiona
           CALL   led        ; voy a la etiqueta "led"
           GOTO   test       ; sino me quedo esperando
led        BTFSC  cont,0     ; si el contador está a 1
           GOTO   on_led     ; lo atiendo en "on_led"
           BCF    PORTB,0    ; sino, apago el LED
           BSF    cont,0     ; pongo el contador a 1
           GOTO   libre      ; y espero que suelten el pulsador
on_led     BSF    PORTB,0    ; enciendo el LED
           CLRF   cont       ; pongo el contador a 0
libre      BTFSC  PORTA,1    ; si no sueltan el pulsador
           GOTO   libre      ; me quedaré a esperar
           RETURN            ; si lo sueltan regreso
                             ; a testear nuevamente
;----------------------------------------
           END
;----------------------------------------
```

El registro que agregué, como te habrás dado cuenta es **cont** y su dirección respectiva **0x0A**. De la configuración de puertos ni hablar, vamos por lo que está en rojo.

```
        CLRF    PORTB       ; limpio el puerto B
        CLRF    cont        ; limpio el contador
        BSF     cont,0      ; pongo el contador a 1
;---------------------------------------
```

CLRF es borrar, o limpiar, o poner a cero, en este caso pongo a cero todo el puerto B y también el registro **cont**, y luego pongo a 1 el primer bit de este último, es decir el bit **0**.

Con esto me aseguro de que no hay ninguna señal en el puerto B, y que el registro **cont** es igual a **0000001**, (**señal de que el LED está apagado**)

Sigamos...

```
;---------------------------------------
test    BTFSC   PORTA,1         ; si alguien presiona
        CALL    led             ; voy a la etiqueta "led"
        GOTO    test            ; sino me quedo esperando
```

Con **BTFSC** estoy probando el segundo bit (**Bit 1**) del puerto A. Si este bit esta a cero es por que nadie lo presionó, entonces salto una línea, y me encuentro con **GOTO test**, así que aquí estaré dando vueltas un buen rato, hasta que a alguien se le ocurra presionar el dichoso pulsador...

Ohhhhhhhhh...!!!, parece ser que alguien lo presionó. Bueno, esta vez no iré a GOTO, sino a **CALL led**, esto es una llamada a la subrutina **led**, allí vamos...

```
led     BTFSC   cont,0      ; si el contador está a 1
        GOTO    on_led      ; lo atiendo en "on_led"
        BCF     PORTB,0             ; sino, apago el LED
        BSF     cont,0      ; pongo el contador a 1
        GOTO    libre       ; y espero que suelten el pulsador

on_led  BSF     PORTB,0             ; enciendo el LED
        CLRF    cont        ; pongo el contador a 0
libre   BTFSC   PORTA,1             ; si no sueltan el pulsador
        GOTO    libre       ; me quedaré a esperar

        RETURN                      ; si lo sueltan regreso
                            ; a testear nuevamente
```

Antes de hacer algo debo saber si el LED está encendido o apagado. Recuerda que si está apagado **cont=0000001**, de lo contrario **cont=00000000**

Pregunta...(**BTFSC cont,0 ?**) - el primer bit del registro cont es igual a uno?...

Si es así el LED está apagado así que lo atenderé en **"on_led"** ahí pondré a uno el primer bit del puerto B (encenderé el led), luego haré **cont=0000000** para saber que desde este momento el LED está encendido.

El tema es que nunca falta aquellos que presionan un pulsador y luego no lo quieren soltar, así que le daremos para que tengan..., y nos quedaremos en la subrrutina **"libre"** hasta que lo suelten, y cuando lo liberen, saltaremos una línea hasta la instrucción **RETURN**.

Así es que caeremos en (**GOTO test**) y esperamos a que opriman nuevamente el pulsador. y si quieres saber si esto funciona ponle el dedito, y caerás otra vez en la subrrutina **"led"**

Pregunta...(**BTFSC cont,0 ?**) - el primer bit del registro cont es 1?...

Noooooooo...!!! Eso significa que el LED esta encendido, entonces lo apagaré (**BCF PORTB,0**), haré **cont=00000001** (de ahora en más LED apagado) y me quedaré en **"libre"** esperando a que sueltes el pulsador, y cuando lo hagas mi 16F84 estará listo para un nuevo ciclo.

Te gustó...?, fue fácil verdad...?

Creo que es un buen punto de partida. En breve hablaremos del famoso MPLAB de Microchip, de momento, trata de conseguirlo, es totalmente gratis desde la web de Microchip.

Set de Instrucciones

Estructura

El PIC 16F84 es uno de los microcontroladores más populares del mercado actual, ideal para principiantes, debido a su arquitectura de 8 bits, 18 pines, y un set de instrucciones RISC muy amigable para memorizar y fácil de entender, internamente consta de:

- Memoria Flash de programa (1K x 14).
- Memoria EEPROM de datos (64 x 8).
- Memoria RAM (68 registros x 8).
- Un temporizador/contador (timer de 8 bits).
- Un divisor de frecuencia.
- Varios puertos de entrada-salida (13 pines en dos puertos, 5 en el A y 8 en el B).

Otras características son:

- Manejo de interrupciones (de 4 fuentes).
- Perro guardián (watchdog).
- Frecuencia de reloj externa máxima 10MHz. (Hasta 20MHz en nuevas versiones). La frecuencia de reloj interna es un cuarto de la externa, lo que significa que con un reloj de 20Mhz, el reloj interno sería de 5Mhz y así pues se ejecutan 5 Millones de Instrucciones por Segundo (5 MIPS)
- No posee conversores analógicos-digital ni digital-analógicos.
- Pipe-line de 2 etapas, 1 para búsqueda de instrucción y otra para la ejecución de la instrucción (los saltos ocupan un ciclo más).
- Repertorio de instrucciones reducido (RISC), con tan solo 30 instrucciones distintas.
- 4 tipos distintos de instrucciones, orientadas a byte, orientadas a bit, operación entre registros, de salto.

Set de Instrucciones del PIC16F84

Para entender mejor cada instrucción se explica a continuación el significado de algunos parámetros:

- f: Registro al que afecta la instrucción
- W: Acumulador (Working register)
- b: Número de bit (hay instrucciones que afectan a un solo bit)
- k: constante (un número)
- d: selección de destino del resultado de la instrucción, puede ser "0" o "1", si es "0" el resultado se guarda en el acumulador (W) y si es "1" se guarda en el registro f al que afecta la instrucción.

Instrucciones orientadas a registros:

- ADDWF f,d Suma W y el registro f, el resultado lo guarda según d (si d=0 se guarda en W y si d=1 se guarda en f).
- ANDWF f,d Realiza la operación AND lógica entre W y f, el resultado lo guarda según d.
- CLRF f Borra el registro f (pone todos sus bits a cero).
- CLRW - Borra el acumulador.
- COMF f,d Calcula el complementario del registro f (los bits que estan a "0" los pone a "1" y viceversa. Resultado según d.
- DECF f,d Decrementa f en uno (le resta uno). Resultado según d.
- DECFSZ f,d Decrementa f y se salta la siguiente instrucción si el resultado es cero. Resultado según d.
- INCF f,d Incrementa f en uno (le suma uno). Resultado según d.
- INCFSZ f,d Incrementa f y se salta la siguiente instrucción si el resultado es cero (cuando se desborda un registro vuelve al valor 00h). Resultado según d.
- IORWF f,d Realiza la operación lógica OR entre W y f. Resultado según d.
- MOVF f,d Mueve el contenido del registro f a W si d=0 (si d=1 lo vuelve a poner en el mismo registro)
- MOVWF f mueve el valor de W a f. Por ejemplo, si queremos copiar el valor del registro "REG1" al registro "REG2" (ya veremos como ponerles nombres a los registros) escribiremos:

  ```
  MOVF REG1,0 ;mueve el valor de REG1 a W
  MOVWF REG2 ;mueve el valor de W a REG2
  ```

  Lo que va después del ; son comentarios
- NOP - No hace nada, solo pierde el tiempo durante un ciclo.
- RLF f,d Rota el registro f hacia la izquierda a través del bit CARRY (todos los bits se mueven un lugar hacia la izquierda, el bit 7 de f pasa al CARRY y el bit CARRY pasa al bit 0 de f). Resultado según d.
- RRF f,d Lo mismo que RLF pero hacia la derecha.
- SUBWF f,d Resta f y W (f - W). Resultado según d.
- SWAPF f,d intercambia los 4 primeros bit de f por los otros cuatro. Resultado según d.
- XORWF f,d Realiza la operación lógica XOR (OR exclusiva) entre W y f. Resultado según d.

Instrucciones orientadas a bits:

- BCF f,b Pone a "0" el bit b del registro f
- BSF f,d Pone a "1" el bit b del registro f
- BTFSC f,b Se salta la siguiente instrucción si el bit b del registro f es "0"

- BTFSS f,b Se salta la siguiente instrucción si el bit b del registro f es "1"

Instrucciones orientadas a constantes y de control:

- ADDLW k Le suma el valor k al acumulador (W).
- ANDLW k Operación lógica AND entre W y el valor k (resultado en W).
- CALL k Llamada a subrutina cuyo inicio esta en la dirección k
- CLRWDT - Borra el registro Watchdog
- GOTO k Salta a la dirección k de programa.
- IORLW k Operación lógica OR entre W y el valor k (resultado en W)
- MOVLW k carga el acumulador con el valor k. Por ejemplo, si queremos cargar el valor 2Ah en el registro "REG1" escribiremos:

  MOVLW 2AH ;carga el acumulador con el valor 2Ah
  MOVWF REG1 ;mueve el valor de W a "REG1"

- RETFIE - Instrucción para volver de la interrupción
- RETLW k carga el valor k en W y vuelve de la interrupción
- RETURN - vuelve de una subrutina.
- SLEEP - El pic pasa a modo de Standby

Más herramientas - Programadores:

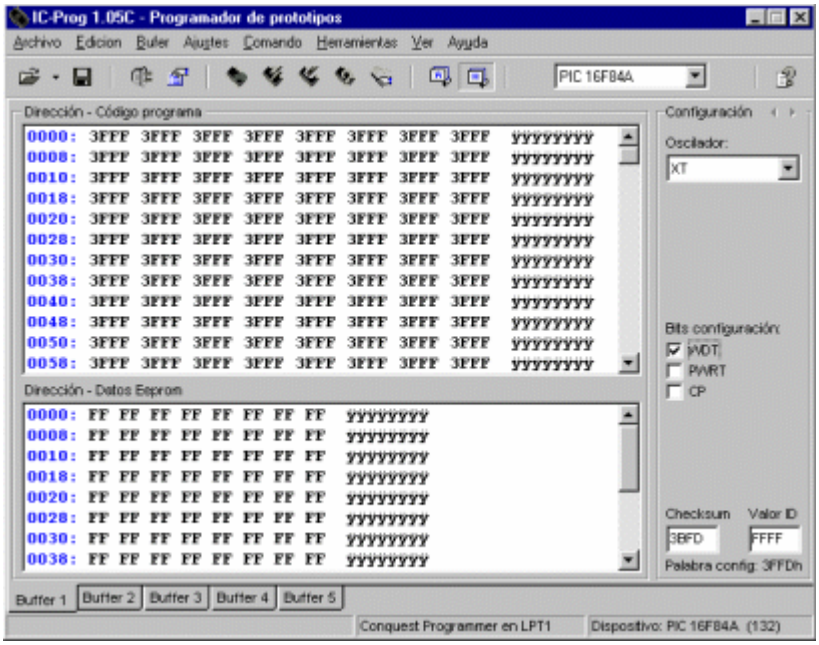
Hay muchos programadores dando vueltas por la Red, uno de los que me llamó la atención fue el Programador PIPO2 de José Manuel García, por su sencillez, por la cantidad de PIC's que soporta, por que no requiere fuente de alimentación, su bajo costo, y por supuesto, por que está listo para montar, jejeje.

El programador anterior no es profesional pero para lo que queremos hacer, se lleva todas las de ganar. Si quieres algo más, como un programador semi-profesional, está su pariente el Programador PP2, también de José Manuel García, aunque requiere unos cuantos euros más, que no creo sea la gran diferencia, por cierto...

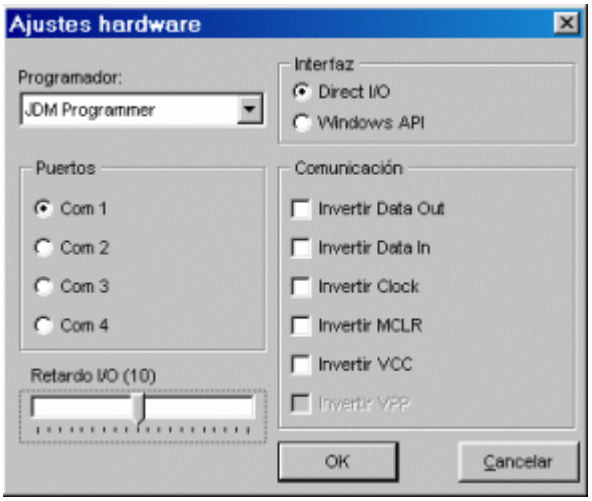
El autor, recomienda su utilización con el IC-Prog, un programa bien realizado en modo windows, es más puedes seleccionar el Idioma, pero para utilizarlo con el programador debes hacerle algunos ajustes. Bueno el mismo autor te indica cómo.

Otra de las características de este programa es que puedes desensamblar un archivo .hex y ver su código fuente, para luego modificarlo o hacer lo que quieras con él. **ojo, respeta los derechos de autor, ok?** :-P

Aquí tienes una imagen de su entorno de trabajo



Y aquí el cuadro de diálogo en que debes realizar los ajustes para el hardware que estés utilizando.



Eso por ahora, luego veremos más herramientas...

Interrupciones y temporizadores

Breve Introducción

Esta vez vamos a tocar uno de los temas que más temía cuando me estaba iniciando con los micros, y aunque parecía difícil, en realidad no lo es tanto.

Trabajar con interrupciones tiene sus ventajas y es hora de aprender a aprovecharlas, si las tenemos dentro del PIC, porqué no usarlas ?

Veamos de qué se tratará todo esto.

Comenzaremos con Interrupciones y analizaremos un poco el Registro INTCON y todos sus Bit's luego nos meteremos un poquito más allá y estudiaremos el registro OPTION, y así aprenderemos a hacer temporizaciones, pero lo haremos de dos formas, la que me gusta y también la otra...

Finalmente hablaremos de un proyecto que cierta vez me encomendaron y al cual le di solución con algunas de las cosas que verás en esta actualización, de acuerdo?

Interrupciones

Una de las características más importante de los microcontroladores y que mencionamos al inicio en nuestro primer tutorial, es que tienen la posibilidad de manejar interrupciones, y qué es esto?

Muy sencillo, se trata de un acontecimiento que hace que el micro deje de lado lo que se encuentra realizando, atienda ese suceso y luego regrese y continúe con lo suyo.

Pues eso son las interrupciones, pero veamos, hay dos tipos de interrupciones posibles, una es mediante una acción externa (es decir por la activación de uno de sus pines), la otra es interna (por ejemplo cuando ocurre el desbordamiento de uno de sus registros)

Fuentes de interrupción en el PCI16F84

En el PIC 16f84 hay 4 fuentes de interrupciones, veamos cuales son...

- Por el pin RB0/INT, que regresa al PIC del modo SLEEP (interrupción externa).
- Por los pines RB4 a RB7, configurados como entrada y en caso de que alguno de ellos cambie de estado (interrupción externa).
- Por desbordamiento del registro TMR0, cuando este registro pasa de 255 a 0 en decimal ó 0xFF a 0x00 en hexa (interrupción interna).
- Al completar la escritura de la EEPROM de datos (interrupción interna).

El tema es que, debe haber algo que nos indique la fuente de interrupción que se ha producido, y estas son las banderas de interrupciones, cada interrupción tiene su propia bandera y es un bit del registro INTCON, que cambia de estado de **0** a **1** cuando se produce la interrupción, salvo la última que se encuentra en el registro EECON1

Ahora veamos cuales son esas banderas...

- Para **RB0/INT** la bandera es **INTF** (Bit1)
- Parar los pines **RB4** a **RB7**, es **RBIF** (Bit0)

- Para **TMR0**, es **T0IF** (Bit2)
- Para la **EEPROM**, es **EEIF** (Bit4) en el registro **EECON1**.

Si prestas atención, todas estas banderas terminan en **F** es decir **FLAG's**.

Bien, ahora veremos todo esto en el registro INTCON, aquí va...

REGISTRO INTCON							
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

El Bit GIE habilita todas las interrupciones, Los Bit's de fondo gris son las banderas, y los BIT's que se corresponden con cada flag son la habilitación de la fuente de interrupción para que esta cambie, recuerda que el flag de EEIE se encuentra en el registro EECON1.

Y ahora los detalles de cada Bit del registro INTCON.

BIT's	L/ E	Reset	Descripción
Bit 7: **GIE** Habilitación General.	L/E	0	1 = Todas las Interrupciones activadas 0 = Todas las Interrupciones desactivadas
Bit 6: **EEIE** Int. de Periféricos	L/E	0	1 = Activada 0 = Desactivada
Bit 5: **T0IE** Int. del TMR0	L/E	0	1 = Activada 0 = Desactivada
Bit 4: **INTE** Int. Externa	L/E	0	1 = Activada 0 = Desactivada
Bit 3: **RBIE** Int. por PORTB	L/E	0	1 = Activada 0 = Desactivada
Bit 2: **T0IF** Bandera del TMR0.	L/E	0	1 = TMR0 desbordado. 0 = No se ha desbordado
Bit 1: **INTF** Bandera - RB0/INT	L/E	0	1 = Ocurrió una interrupción externa 0 = No ha ocurrido interrupción externa
Bit 0: **RBIF** Bandera - RB4:RB7	L/E	x	1 = Al menos un pin cambio de estado 0 = Ningún pin ha cambiado de estado.

En la tabla, los valores de **L** ó **E** son para que sepas si el bit es de lectura o escritura, los valores de **Reset** son el estado de cada Bit después de producirse un reset o cuando se inicia el micro.

Por cierto y antes de que lo olvide, si bien cada flag cambia o se pone a 1 al producirse una interrupción, es tarea tuya borrarlo o ponerlo a cero nuevamente, ya que si no lo haces el micro estará siempre interrumpido o lo que es lo mismo, creerá que la interrupción se está produciendo continuamente.

Recapitulemos un poco... Ya sabemos como y cuando se produce una interrupción, conocemos las banderas que nos indican la fuente de interrupción producida, conocemos el registro donde se encuentran y los valores que toman cada uno de sus BIT's. Pues bien, ahora hay que atenderlas.

Lo primero que debes saber, es que cuando una interrupción se produce, sea cual fuere la fuente de interrupción, el micro deja todo y salta a la dirección 0x04, éste es el vector de interrupción, si recuerdas de nuestro primer tutorial, siempre saltábamos por encima de esta dirección para iniciar nuestro programa, en esta dirección es donde escribiremos la rutina que dé servicio a todas las

interrupciones, o bien haremos un salto a donde se encuentre ese trozo de código, el cual se conoce como **ISR** (Rutina de Servicio de Interrupción)

El Tiempo de Procesamiento de la ISR debe ser lo más breve posible, para dar lugar a que se ejecuten las otras interrupciones, ya que puedes haber habilitado más de una de ellas.

Lo más crítico de una interrupción es tener que guardar todos los registros importantes con sus respectivos valores, para luego restaurarlos, y así el micro pueda continuar con la tarea que estaba realizando cuando fue interrumpido.

Rutina de Servicio de Interrupciones (ISR)

La tarea de guardar todos los registros importantes puede ser mas o menos complicada si el programa que estás realizando es demasiado extenso o principalmente cuando en la ISR modificas alguno de los valores de esos registros, en algunos casos no es necesario ya que por lo general se trata de no modificarlos utilizando registros alternativos, pero veamos como hacerlo.

Primero debes guardar el contenido del registro W, el problema de mover W a otro registro (haciendo uso de **MOVF**) es que esta instrucción corrompe la bandera Z, modificando el registro de Estado. Según la hoja de datos otorgada por Microchip, en uno de sus apartados recomienda una secuencia de código que permite guardar y restaurar los registros sin modificarlos.

Suponte que W=0x0A y ESTADO=0xAF, La forma de guardarlos y recuperar estos registros sería la siguiente:

```
; ======== Inicio - Rutina de Servicio de Interrupción =========
; ========== Guardando W y el Registro de Estado ==========

MOVWF   Reg_W           ; Guardamos W en Reg_W (Reg_W=0x0A)

SWAPF   ESTADO,W        ; invertimos los nibbles del registro ESTADO
                        ; y lo pasamos a W (ESTADO=0xAF), (W=0xFA)
MOVWF   Reg_S           ; Guardamos el contenido de ESTADO (Reg_S=0xFA)
  .         .
  .         .           ; Atendemos la interrupción
  .         .

; ======== Fin - Rutina de Servicio de Interrupción ===========
; ========  Restaurando W y el Registro de Estado ===========

SWAPF   Reg_S,W         ; invertimos los nibbles de Reg_S
                        ; y lo pasamos a W (Reg_S=0xFA), (W=0xAF)
MOVWF   ESTADO          ; Restauramos ESTADO (ESTADO=0xAF)
SWAPF   Reg_W,f         ; invertimos los nibbles de Reg_W (Reg_W=0xA0)
SWAPF   Reg_W,W         ; y lo pasamos a w invirtiéndoles nuevamente
                        ; Ahora W=0x0A
RETFIE
```

Reg_W y **Reg_S** son registros alternativos para guardar los valores del registro W y del registro de estado respectivamente.

SWAPF ESTADO,W

Es como decirle "invierte los nibbles del registro ESTADO y guárdalos en W".

La instrucción SWAPF invierte los nibbles del registro, por ejemplo si el registro tenia 0xAF luego de SWAPF quedará 0xFA, si especificas **W** el valor invertido se guarda en W si indicas **f** se guardará en el mismo registro, así...

SWAPF Reg_W,f

Es como decirle "invierte los nibbles de Reg_W y guárdalos en Reg_W".

Creo que se entiende...

Bien, lo bueno de utilizar la instrucción SWAPF en lugar de MOVF es que no afecta la bandera Z del registro de ESTADO, y aunque los nibbles se invierten, al restaurarlos los vuelves a invertir nuevamente para dejarlos como estaban.

Como dije anteriormente, no siempre será necesario, todo depende del programa que estés haciendo.

Algo que no mencioné, es la instrucción...

RETFIE

Veamos y tratemos de resumir un poco lo que vimos hasta ahora...

Si se ha producido una interrupción, obviamente una de las banderas del registro INTCON cambiará de estado y el micro irá a la dirección 0x04 como si se hubiera producido un CALL (una llamada) a esa dirección para ejecutar la ISR, por lo tanto la pila o STACK se carga una posición más, y el mecanismo de las interrupciones se deshabilita (es decir **GIE=0**) para dar lugar a la ISR.

Ahora bien, debes recuperar los registros importantes (lo que acabamos de ver), averiguar la fuente de interrupción, atender la interrupción, luego restaurar aquellos registros importantes, reponer el estado de las banderas del registro INTCON (aquellas que fueron modificadas por la interrupción) y regresar, pero no con un RETURN, ya que no estas regresando de una rutina cualquiera, sino de una interrupción, la cual está deshabilitada, y para habilitarla nuevamente es recomendable utilizar la instrucción **RETFIE**, y yo cumplí...!!!

Y ahora todas las interrupciones están habilitadas nuevamente, es decir **GIE=1**

Nada impide que utilices **RETURN** pero deberás usar una instrucción más para habilitar GIE si deseas continuar usando la interrupción, esto queda a criterio del programador, Microchip recomienda el uso de **RETFIE** y yo como chico bueno la utilizo.

Como era de esperarse no todo termina aquí, ya que algunos de los parámetros para las interrupciones se encuentran en otro registro, el registro OPTION, veamos de que se trata...

El Registro OPTION

Este es otro de los registros que tienen mucho que ver con las interrupciones, algunos de sus Bit's deben ser modificados, según la aplicación que estés realizando.

Por ejemplo; dijimos que por el pin RB0/INT, regresas al PIC del modo SLEEP, lo cual podría hacerse mediante un pulsador, suponte que el pulsador está al polo positivo (VCC) y con una resistencia a GND, de tal modo que la interrupción se produzca al enviar un 1 (presionando el pulsador), pero también podría hacerse enviando un 0 (liberando al pulsador). por lo tanto la interrupción debe ser **sensible a** un **1** o bien a un **0**, como sabrá esto el micro...???, pues muy fácil, hay que especificarlo, y esto se hace en el Bit6 (**INTDEG**) del registro OPTION, con un **1** será sensible al flanco ascendente, y en el momento que envíes un 1 por el pulsador se producirá la interrupción, si pones ese Bit a **0** será sensible al flanco descendente y la interrupción se producirá cuando liberes el pulsador, es decir enviando un 0.

Este es el registro OPTION

REGISTRO OPTION							
RBPU	INTDEG	T0CS	T0SE	PSA	PS2	PS1	PS0

Y aquí verás como configurar algunos de sus BIT's

BIT's	L / E	Reset	Descripción
Bit 7: **RBPU** Pull-up p' PORTB	L/E	1	1 = Cargas Pull-Up Desconectadas 0 = Cargas Pull-Up Conectadas
Bit 6: **INTEDG** Flanco/Interrup.	L/E	1	1 = RB0/INT será sensible a flanco ascendente 0 = RB0/INT será sensible a flanco descendente
Bit 5: **T0CS** Fte./Reloj p' TMR0	L/E	1	1 = Pulsos por el pin RA4/T0CKI (contador) 0 = Pulsos igual Fosc/4 (temporizador)
Bit 4: **T0SE** Flanco/T0CKI	L/E	1	1 = Incremento TMR0 en flanco descendente 0 = Incremento en flanco ascendente
Bit 3: **PSA** Divisor/Frecuencia	L/E	1	1 = Divisor asignado al WDT 0 = Divisor asignado al TMR0

Como puedes ver, en la tabla no figuran los primeros tres Bit's, y es que la combinación de los BIT's; PS2, PS1 y PS0 (2, 1 y 0 respectivamente) determinan el valor del divisor de frecuencia o prescaler (mmmmmmm, no te preocupes que cuando terminemos con este tutorial o mejor dicho cuando hablemos de temporizaciones sabrás de que se trata)...

Basta de teoría, es hora de pasar a la práctica, haremos nuestro primer programa con interrupciones, que emocionante...!!! ya me estaban picando las manos para codificar...!!!, quería hacer algo complejo e interesante, pero temo que te pierdas y lo que es peor, temo enredarme al tratar de explicarlo, así que haremos algo sencillito, ok? Luego lo iremos complicando y pondremos a llorar a muchos.

Codificando interrupciones

Ya estoy de regreso nuevamente, y a ver quién me sigue... que esta vez haré un programa que a muchos les puede resultar bobo, así que... a no criticar, que ya lo advertí.

Comenzamos?

Bien, el programa consiste en preparar todo para el encendido de un LED que conectaremos en RB1, pero como dije, sólo prepararemos todo, porque luego haremos dormir al micro hasta que interrumpamos su sueño para atender un pulsador conectado a RB0/INT, momento en el cual deberá encender el LED, y regresar a dormir nuevamente, y cuando vuelvas a presionar el pulsador haremos que lo apague y otra vez lo despacharemos a dormir.

Esto ya lo hicimos anteriormente, sólo que ahora lo haremos con interrupciones, de acuerdo?

```
;--------------Encabezado------------

            LIST      P=16F84
            #include  <P16F84.INC>

;-------Configuración de puertos-------

            ORG       0x00
            GOTO      inicio
            ORG       0x04
            GOTO      ISR
            ORG       0X05

inicio      BSF       STATUS,RP0       ; configurando puertos
            MOVLW     0x01             ; carga w con 0000 0001
            MOVWF     TRISB            ; RB0/INT es entrada
            BCF       OPTION_REG,6     ; seleccionamos flanco descendente
            BCF       STATUS,RP0

;-------Habilitación de interrupciones-------

            BSF       INTCON,GIE       ; habilitamos todas las interrupciones
            BSF       INTCON,INTE      ; que sean interrupciones externas
            CLRF      PORTB            ; limpio el puerto B
sueño       SLEEP
            GOTO      sueño            ; Dulces sueños...!!!

;------------Rutina de servicio de interrupciones------------

ISR         BTFSC     PORTB,0          ; verificamos que suelten el pulsador
            GOTO      ISR

            BTFSC     PORTB,1                  ; y ahora sí, si el led está a 1
            GOTO      off_led          ; ire a off_led para apagarlo
            BSF       PORTB,1                  ; sino, enciendo el LED
            BCF       INTCON,INTF      ; borro bandera de interrupción
            RETFIE

off_led     BCF       PORTB,1                  ; apago el LED
            BCF       INTCON,INTF      ; borro bandera de interrupción
            RETFIE

;----------------------------------------
            END
;----------------------------------------
```

Desde nuestros primeros tutoriales hemos alcanzado a conocer varias de las instrucciones que se encuentran en este trozo de código, razón por la cual no las describiré, así es que vamos por aquello que está en rojo...

```
ORG     0x04
GOTO    ISR
```

La primera línea es el vector de interrupción, y cuando ésta se produzca, el código de programa apuntará a esta dirección y continuará con la siguiente instrucción, es decir GOTO ISR, la cual es un salto a ISR (Rutina de Servicio de Interrupciones) para atender dicha interrupción.

Configuramos el puerto B, como habrás notado, hemos configurado RB0/INT como entrada y el resto de los bits como salida, luego...

```
BCF     OPTION_REG,6    seleccionamos flanco descendente
```

En la página anterior dijimos que podíamos seleccionar el flanco con el cual se producirá la interrupción, pues eso es lo que estamos haciendo con esta instrucción, entonces vamos al registro OPTION y ponemos el BIT6 a **"0"** de este modo la interrupción se producirá cuando suelten el pulsador.

Ahora pasamos a lo más interesante, la habilitación de las interrupciones...

```
BSF     INTCON,GIE      ; habilitamos todas las interrupciones
BSF     INTCON,INTE     ; que sean interrupciones externas
```

Observa que la habilitación de interrupciones se hace en el banco0 ya que el Registro INTCON se encuentra en este banco. Bien, En la primera línea hacemos una habilitación general de todas las interrupciones, hacemos GIE=1, en la segunda línea, habilitamos interrupciones externas, hacemos INTE=1, recuerda que la bandera para la interrupción por el pin RB0/INT es INTF, no lo olvides, pues esta cambiará cuando la interrupción se produzca y luego de atenderla deberemos volverla a cero.

Lo que viene ahora es simplemente limpiar el puerto B, y luego...

```
sueño   SLEEP
        GOTO    sueño
```

SLEEP es la instrucción que pone al micro en estado de bajo consumo, es como que todo se detiene y éste pasa a modo de reposo, (consulta el set de instrucciones para mayor detalle...) debí haber puesto simplemente SLEEP pero veamos, si se ejecutara la instrucción SLEEP, el micro entraría en reposo hasta que se produce la interrupción, lo cual dijimos anteriormente que es como una llamada (un call), cuando regrese se encontrará con **GOTO sueño** y lo volveremos a dormir.

Te imaginas? si no pusiéramos el **GOTO sueño,** cuando regrese de la interrupción pasaría a la **ISR** (Rutina de servicio de interrupción), y lo peor de todo, es que lo haría sin que se produzca la interrupción, sería un gran dolor de cabeza...!!!

Pero bueno, ahora nos quedamos a dormir con el micro hasta que algún travieso active el pulsador de RB0/INT ...Felices sueñosssss...!!!!

chico malo...!!!, que siempre me pones a prueba, Ya veo que no pudiste esperar y presionaste el pulsador...

De acuerdo... sabes a donde me llevas? justo a...

```
ORG     0x04
GOTO    ISR
```

Entonces vamos hacia allá, ahora te daré para que tengas, guardes y repartas...

Lo que viene, es la rutina de servicio de interrupción **ISR** y comenzamos con un...

```
ISR   BTFSC   PORTB,0   ; verificamos que suelten el pulsador
GOTO    ISR
```

ISR no es una instrucción, sino la etiqueta que atiende la interrupción (pude haber puesto rut_serv u otra cosa, en fin...). Con **BTFSC PORTB,0**, prevenimos los rebotes, no se si era necesario ya que seleccionamos flanco descendente para este pin, pero por si las moscas lo puse, en realidad suele pasar que cuando se libera el pulsador se genera una pequeña chispa, la cual ya conocemos como rebote eléctrico, sólo lo puse por prevención.

Ahora si atenderemos la interrupción, comenzando por...

```
BTFSC   PORTB,1
```

BTFSC PORTB,1 es probar si el segundo bit (Bit1 de PORTB) está en **0**, es decir si el LED está apagado y saltar un línea si es así.

*En lecciones anteriores utilizamos un registro llamado **cont** con el cual sabíamos si el LED estaba encendido o apagado, y aquí tienes una forma de optimizar ese código, espero que no te pierdas con esto...!!!*

Bien... en el incio, el LED está apagado, por lo tanto saltamos una línea y pasamos a…

```
BSF   PORTB,1
```

es decir hacemos **RB1=1** (prendemos el LED). Perfecto, la interrupción ya fue atendida, pero ahora debemos habilitarla de nuevo así permitimos que se vuelva a ejecutar, y como tenemos un único pulsador el cual me cambió la bandera INTF, deberemos borrarla nuevamente, así es que...

```
BCF   INTCON,INTF   ; borro bandera de interrupción
```

Obviamente al producirse la interrupción se hizo **GIE=0** para darnos lugar a atenderla, entonces...

```
RETFIE
```

y ahora **GIE=1**, las interrupciones están nuevamente habilitadas la bandera de RB0/INT está lista para una nueva interrupción y retornamos a ...

```
sueño   SLEEP
        GOTO    sueño
```

y esperaré a que pulses RB0, pues si ya lo hiciste habrás ido por segunda vez a ...

```
ORG     0x04
GOTO    ISR
```

prevenimos los rebotes y luego vamos a...

BTFSC PORTB,1

es decir, prueba y salta si el Bit1 de PORTB es cero, y como esta vez el LED está prendido... simplemente harás un…

GOTO off_led ; ire a off_led para apagarlo

un salto a la etiqueta off_led...

off_led BCF PORTB,1 ; sino, apago el LED

no se si requiere explicación pero bueno, pones a cero el Bit1 de PORTB. Finalmente...

BCF INTCON,INTF ; borro bandera de interrupción
RETFIE

Ahora estamos listos para comenzar de nuevo... te gustó...???, bieeeenn, aplausos para quien logró comprender... y ahora la pregunta del millón... que pasó con el mapa de memoria...???, donde se definieron las posiciones de memoria para los registros TRISB, PORTB, el registro OPTION..??? y el INTCON...???, donde demonios se definieron estas posiciones de memoria que supuestamente deben estar en el encabezado...???

No me digas que no te percataste de eso...!!!

La respuesta la tendremos más adelante… sigue con esta interesante parte de las interrupciones.

Simulando la interrupción con MPLAB

Antes de comenzar con la simulación vamos a aclarar un pequeño detalle, o mejor dicho... vamos a dar respuesta a la pregunta del millón... no pensarías que te dejaría así...!!!, noooooooo...!!!

En este manual no se incluye ningún apartado sobre el uso de MPLAB, lo tendrás que buscar aparte porque de momento y con el avance de las versiones, lo he dejado para más adelante, así que te recomiendo leer algo sobre MPLAB.

Sigo…..

Bueno, antes de que copies el código en MPLAB e intentes ensamblarlo, lo cual seguramente te dará quinientos mil errores, debes saber que Cuando instalaste MPLAB allá en:

C:\Archivos de programa\MPLAB

Se instaló también un pequeño archivo en el mismo directorio, llamado **P16F84.INC**, Bien, búscalo y una vez lo tengas a mano... lo abres, que le haremos un par de modificaciones...

Te encontrarás con algo como esto...

```
;===================================
;
;       Register Definitions
;
;===================================
```

```
W                   EQU     H'0000'
F                   EQU     H'0001'

;----- Register Files------------------------------------

INDF                EQU     H'0000'
TMR0                EQU     H'0001'
PCL                 EQU     H'0002'
STATUS              EQU     H'0003'
FSR                 EQU     H'0004'
PORTA               EQU     H'0005'
PORTB               EQU     H'0006'
EEDATA              EQU     H'0008'
EEADR               EQU     H'0009'
PCLATH              EQU     H'000A'
INTCON              EQU     H'000B'

OPTION_REG          EQU     H'0081'
TRISA               EQU     H'0085'
TRISB               EQU     H'0086'
EECON1              EQU     H'0088'
EECON2              EQU     H'0089'
```

Bueno, es sólo una parte del archivo P16F84.INC, éste archivo contiene los nombres de los registros con sus respectivas posiciones de memoria, aquello que nosotros veníamos indicando en cada código que íbamos escribiendo, y una que otra cosita más como los nombres de los Bit's de cada uno de los registros, y si recuerdas siempre debíamos indicar la posición 0x05 para TRISA y 0x06 para TRISB, por tanto para OPTION_REG (registro OPTION) sería 0x01, te preguntarás... porque aquí las cosas se ven totalmente distintas...???

Lo que ocurre, es que cuando pasas al banco 1... TRISA está quinto en ese banco, es decir está en la posición 0x05, lo mismo ocurre con TRISB en 0x06, y por ende OPTION_REG está en 0x01, observa ahora los bancos de la RAM de nuestro primer tutorial... y compara con lo que acabamos de ver...

Convencido?.

El tema es que para evitar tener que definirlos, tomaremos aquello que marqué en rojo y lo cambiaremos por...

```
OPTION_REG          EQU     H'0001'
TRISA               EQU     H'0005'
TRISB               EQU     H'0006'
```

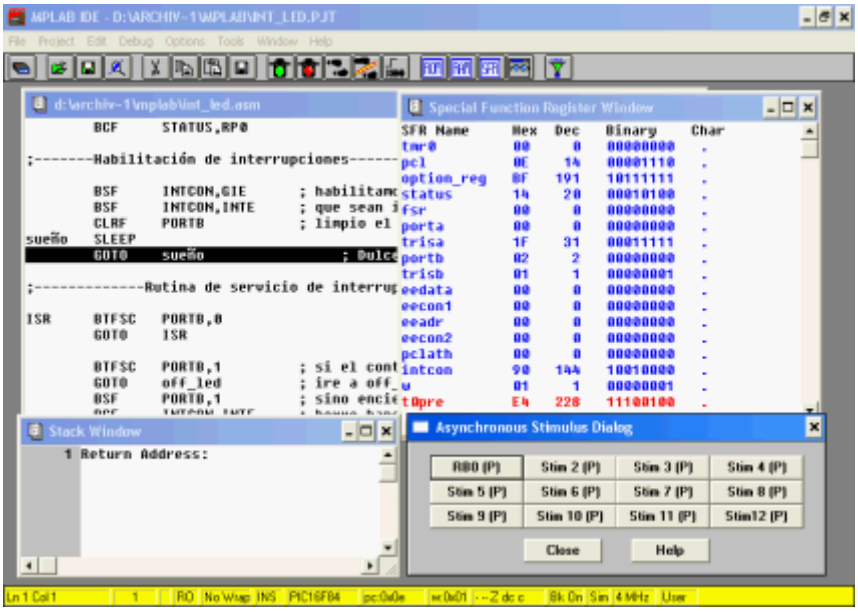
De ahora en más siempre que hagamos un programa será obligación colocar en el encabezado de nuestro código la siguiente línea...

#include <P16F84.INC>

De acuerdo... Ahora guardamos los cambios, cerramos el archivo y comenzamos un nuevo proyecto en MPLAB al cual lo llamaremos INT_LED.pjt, y en él creamos int_led.asm, copias el código, lo pegas y le das a...

Project --> Build All

Y como todo está perfecto...!!! comenzaremos la simulación, te debería quedar algo así...



No te asustes por todas las ventanas abiertas, son sólo 4, y todas accesibles desde el menú Window, la primera es el código, la que está al lado es **Special Function Register** en la que veremos como cambian los registros, la de abajo es la ventana que nos muestra la pila o STACK y la última es la de **Asynchronous Stimulus** esta última se encuentra en el menú **Debug --> Simulator Stimulus**, cuando la abras configura **Stim 1 (P)** como **RB0 (P)**, eso hará que cuando lo presionemos envíe un pulso de nivel alto por el pin RB0, al configurarlo como **(P)** se convierte en un pulsador, ahora sí, ya estamos listos para comenzar...

Reseteamos el Micro o presionamos F6, y te habrás ubicado en **GOTO inicio**, ahora ve a...

Debug --> Run --> Animate

y quedará todo como está en la imagen anterior

Aquí haremos un par de observaciones, fíjate que estas en **GOTO sueño**, ésta es la siguiente instrucción que se debería ejecutar, pero no lo hace ya que el micro está dormido gracias a la instrucción SLEEP, observa también que en la ventana **Special Function Register** todo se pintó de azul por tanto el micro se detuvo y apagó casi todo. El STACK está vacío ya que no se produjo ninguna llamada, PORTB está en 00000000, es decir que el LED está apagado (RB1=0) y no hay ninguna interrupción todavía (RB0=0), finalmente échale una mirada al registro INTCON que esta en 10010000 es decir GIE=1 e INTE=1 las interrupciones están habilitadas.

Ahora viene lo bueno...

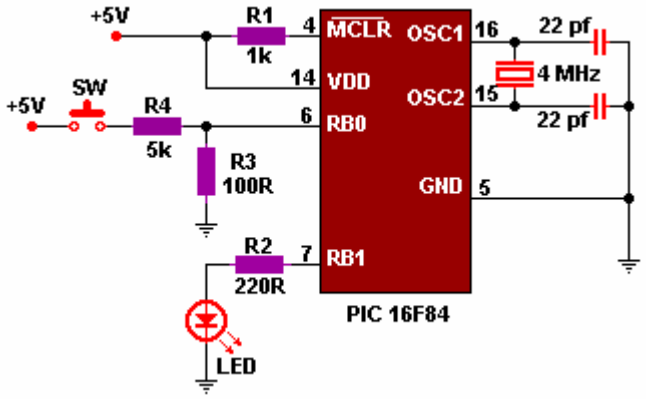
Envía un pulso por **RB0 (P)**, y verás que la interrupción hace saltar al micro en la dirección 0x04, (no esperes ver en PORTB que RB0 se ponga a 1 ya que al configurar RB0 (P) sólo envía un pulso momentáneo el cual es difícil notar), el STACK se incrementa en una posición, y en el registro INTCON se deshabilita GIE, la bandera INTF se pone a 1, luego el micro apunta a ISR, atiende la interrupción encendiendo el LED (RB1=1), luego Borra la bandera INTF y con RETFIE vacía la pila habilitando GIE nuevamente para regresar a GOTO sueño donde ejecutará SLEEP, para dormirse y esperar a que presiones nuevamente el pulsador...

Bravooooo...!!!

Tienes mucho para entretenerte, una jugada que te pondrá de pelos...

Observa lo que harás; configura **RB0 (P)** como **RB0 (T)**, resetea el micro y comienza nuevamente, la mayor sorpresa es que cuando lo presiones, RB0 se pondrá a 1 y no habrá interrupción, esto es así por que seleccionamos flanco de bajada para la interrupción en RB0/INT, aquello que hicimos en el Bit6 del registro OPTION, recuerdas eso...???, ok... entonces debería haber interrupción cuando presiones nuevamente RB0 y ya no diré más...

Bueno... sólo dejarte el circuito para que lo pruebes cuando grabes el programa en el PIC...



Suerte...!!!, y a no bajar los brazos que lo probé y funciona perfectamente.

Por si te quedan dudas de como funciona el programa, realiza la simulación en modo STEP (con el botón de los zapatitos), eso debería responder a todas tus dudas.

Interrupciones Internas y Temporizaciones

Itroducción

Lo que hicimos anteriormente fue trabajar con interrupciones externas aunque no tocamos aquello de las interrupciones por los pines RB4 a RB7, el procedimiento es muy similar sólo debes tener en cuenta la bandera que informa el estado de las interrupciones y estudiarlas para saber por cual de esos pines se produjo.

Lo que veremos ahora será como trabajar con una interrupción interna, algo no muy distinto a lo anterior pero que lo vale, ya que haremos uso de algo que fue pedido en el foro por César Bonavides, a quien envío mis cordiales saludos allá en méxico.

Antes de entrar en materia, debo confesar que lo fuerte de esta sección serán las **Temporizaciones**, aunque obviamente haremos uso de interrupciones, de acuerdo...???

Desde mis inicios con los microcontroladores, temporizar fue algo que me trajo grandes dolores de cabeza, y aunque me las arreglaba como podía, me quedaba siempre con la duda, *que hago cuando quiero temporizar sólo un segundo...???*, supongo que a muchos les suele pasar, hasta que finalmente no lo pude evitar y comencé a escarbar en cuanto tutorial caía en mis manos, y de

ello surgió lo que describiré aquí, y trataré de explicarlo lo mejor posible, espero no meter la pata...!!!

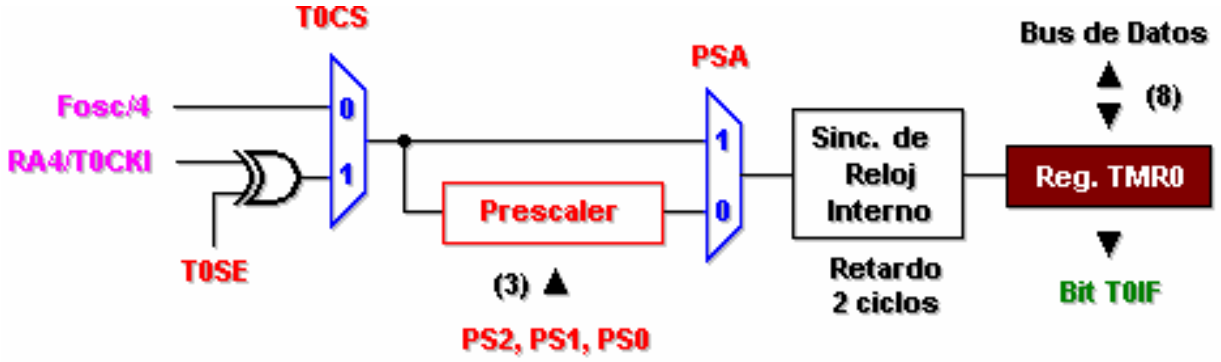
Como siempre... un poco de teoría no viene mal no crees...???

Temporizaciones

Algunas de las cosas que deberemos tener en cuenta son...

Por un lado, el oscilador externo, y yo lo haré con un XT de 4 Mhz es decir 4 millones de ciclos por segundo, y por otro lado un registro llamado TMR0, este registro es un Temporizador/Contador de 8 bits, que como es lógico cuenta en binario y en forma ascendente es decir... de 0x00 a 0xFF, lo interesante de este registro es que cuando ocurre un desbordamiento, es decir pasa de 0xFF a 0x00 hecha sus alaridos en el registro INTCON modificando la bandera T0IF, es decir, produce una interrupción.

Veamos ahora como está conformada la estructura del micro para trabajar con este registro, y quienes están involucrados en ello.



Creo que ahora se van a entender un poco mejor las cosas...

Aquello que está en rojo son Bit's configurables en el Registro OPTION, lo que está en verde **Bit T0IF** es el Bit2 del registro INTCON (la bandera que se altera cuando el TMR0 se desborda), y lo que está en rosado... bueno, hablemos de ello...

El registro TMR0 puede trabajar de dos modos distintos, ya sea como temporizador o bien como contador el cual seleccionarás en **T0CS** (Bit5 del registro OPTION)

En modo Temporizador: El TMR0 se incrementa con cada **Ciclo de Instrucción** (**Fosc/4**) y cuando el registro se desborde provocará una interrupción.

En modo Contador: El TMR0 se incrementará con cada pulso que ingrese por el pin **RA4/T0CKI**, y por supuesto... cuando se desborde producirá la interrupción. **T0SE** es el Bit4 del Registro OPTION, en él seleccionas el flanco con el cual se incrementará el TMR0 cuando haya un pulso por RA4.

Algo que quería mencionar es que el micro dispone de dos temporizadores, el TMR0 y WDT (Watchdog). El primero es el que estamos tratando en esta sección, el segundo es el *perro guardián*, lo que hace es vigilar cada cierto tiempo que el micro no se quede colgado, por ejemplo cuando se queda detenido en un bucle infinito o en una larga espera de un acontecimiento que no se produce, entonces actúa reseteando al micro, en otro momento hablaremos más de él...

El registro OPTION y el prescaler

El **Prescaler** es un predivisor de frecuencia que se utiliza comúnmente para programar tiempos largos y puedes aplicarlo al TMR0 o al WDT, esto lo configuras en **PSA** Bit3 del registro OPTION.

Hablamos tanto del Registro OPTION que decidí traértelo nuevamente pero marcando con color aquello que acabamos de mencionar, y aquí está...

REGISTRO OPTION							
RBPU	INTDEG	T0CS	T0SE	PSA	PS2	PS1	PS0

Y para refrescar...

Bit 5: **T0CS**: Selecciona la fuente de Reloj para TMR0
1 = Pulsos por el pin RA4/T0CKI (contador)
0 = Ciclo de instrucción interno (temporizador)

Bit 4: **T0SE**: Flanco de incremento para RA4/T0CKI
1 = Incrementa TMR0 en flanco descendente
0 = Incremento en flanco ascendente

Bit 3: **PSA**: Bit de asignación del Prescaler
1 = Divisor asignado al WDT
0 = Divisor asignado al TMR0

Bit 2-0: **PS2, PS1, PS0**: Selección del prescaler (divisor de frecuencia)

El prescaler debe tener algún valor, y si bien no puedes cargarle un valor cualquiera, tienes la posibilidad de seleccionarlo según la combinación de **PS2, PS1** y **PS0**.

En la siguiente tabla puedes ver estos posibles valores...

PS2	PS1	PS0	División del TMR0	División del WDT
0	0	0	1/2	1/1
0	0	1	1/4	1/2
0	1	0	1/8	1/4
0	1	1	1/16	1/8
1	0	0	1/32	1/16
1	0	1	1/64	1/32
1	1	0	1/128	1/64
1	1	1	1/256	1/128

Algo que me estaba quedando pendiente es la **Sincronización de Reloj interno**, por lo visto, según la hoja de datos, el micro genera un pequeño retraso de 2 ciclos de instrucción mientras sincroniza el prescaler con el reloj interno.

Bien, ahora veremos para que nos sirve toda esta información...

Cálculo de temporizaciones con el registro TMR0

El tiempo empleado en una temporización se puede calcular a partir de **un ciclo de instrucción** (es decir 1 instrucción por cada microsegundo, si estas trabajando con un XT de 4 Mhz), también

necesitas el valor del **Divisor de Frecuencia** (el que seleccionabas con los Bit's PS2, PS1 y PS0), y finalmente con **el complemento del valor cargado en TMR0** (es decir 255-TMR0), la ecuación que te permite realizar el cálculo es la que sigue...

Temporización = Ciclo de instrucción * (255-TMR0) * Divisor de Frecuencia

Veamos un ejemplo:

Supongamos que deseas una temporización de 10 ms (10 milisegundos), que estás trabajando con un XT de 4 Mhz, y que además seleccionaste como Divisor de frecuencia 256 (es decir PS2,PS1,PS0 = 1,1,1).

Pregunta; Cuál es el valor que se debe cargar en TMR0?

Lo arreglaremos con un pasaje de términos...

255-TMR0 = Temporización(en microsegundos)/(1 ciclo/us * Div. de Frec.)

y reemplazando tendrás...

255-TMR0 = 10000 us/(1 ciclo/us * 256)
255-TMR0 = 10000 /(256 ciclos)
255-TMR0 = 39,0625 ciclos
255-TMR0 ~ 39 ciclos

Eso significa que en TMR0 deberás cargar 255-39=**216** (0xD8 en hexa) y a partir de allí el TMR0 contará los 39 ciclos que faltan para desbordarse y producir la interrupción, y el tiempo que tardará en hacerlo es aproximadamente 10000 us, o sea 10 ms.

Antes de seguir, despejemos un par de dudas:

1 seg. = 1000 ms = 1000000 us y ...

1 ciclos/us es el tiempo empleado en ejecutarse una instrucción

ok..., sería bueno que confirmes si la mayor temporización que se puede obtener haciendo uso de este registro es 0,06528 segundos, será cierto...??? ahí queda...!!!

Lo que haremos ahora, será codificar el ejemplo visto anteriormente, pero una vez producida la interrupción encendemos un LED, luego volvemos, temporizamos 10 ms y en la próxima interrupción, lo apagamos, es decir, el LED parpadeará cada 10 ms, como es obvio, no lo vamos a notar, así que sólo lo simularemos en MPLAB, (*en realidad si se nota, luego te cuento como*).

Bien, el código es el siguiente:

```
;--------------Encabezado------------

            LIST      P=16F84
            #include  <P16F84.INC>

;-------Configuración de puertos-------

            ORG       0x00
            GOTO      inicio

            ORG       0X04              ; Atiendo la interrupción
            BTFSS     PORTB,0           ; si el LED está apagado
            GOTO      LED               ; voy a LED y lo enciendo
            BCF       PORTB,0           ; sino apago el LED
            BCF       INTCON,2          ; limpio la bandera T0IF
            RETFIE                      ; regreso habilitando la interrupción
LED         BSF       PORTB,0           ; enciendo el LED
            BCF       INTCON,2          ; borro la bandera T0IF
            RETFIE                      ; regreso habilitando la interrupción

inicio      BSF       STATUS,5          ; configurando puertos
            CLRF      TRISB             ; puerto B es salida
            MOVLW     0x07              ; cargo w con 00000111
            MOVWF     OPTION_REG        ; el Divisor = 256
            BCF       STATUS,5

            MOVLW     0XA0              ; cargo w con 10100000
            MOVWF     INTCON            ; habilitamos GIE y T0IE
            CLRF      PORTB             ; limpiamos PORTB

tiempo      MOVLW     0XD8              ; cargo w con 216
            MOVWF     TMR0              ; lo paso a TMR0
NADA        BTFSC     TMR0,7            ; me quedo haciendo nada
            GOTO      NADA              ; hasta que TMR0 desborde, y entonces
            GOTO      tiempo            ; volveré a cargar TMR0

;-----------------------------------------
            END
;-----------------------------------------
```

Aquí vamos...

ORG 0X04 ; Atiendo la interrupción

Aquí vendremos cuando se desborde el TMR0, es decir cuando se produzca la interrupción y no haremos una ISR aparte como lo hicimos anteriormente, atenderemos la interrupción directamente aquí.

El código que sigue es como dice el comentario, se trata de verificar si RB0 está a 1 (es decir si el LED esta encendido), y como de comienzo no lo está, irá a GOTO LED, ahí lo enciende, luego...

```
BCF   INTCON,2   ; limpio la bandera T0IF
```

Esto es lo que debemos tener en cuenta para salir de una interrupción, borrar la bandera que indica al micro que hubo una interrupción, o nos quedaremos siempre en la rutina de servicio. Finalmente con...

```
RETFIE
```

habilitamos nuevamente la interrupción.

Pasemos ahora a la etiqueta **inicio**, lo primero que haremos será cambiar de banco y luego configurar el puerto B como salida, y aquí viene lo nuevo...

```
MOVLW   0x07            ; cargo w con 00000111
MOVWF   OPTION_REG   ; el Divisor = 256
```

Veamos que Bit's estamos configurando en **OPTION_REG**

Los Bit's 7 y 6 no los utilizamos por ahora, T0CS=0 (**TMR0 es temporizador**), T0SE=0 (no se usa), PSA=0 (**Prescaler asignado a TMR0**), PS2,PS1,PS0=1,1,1 (**Prescaler es 256**), en conclusión 00000111=0x07 y es lo que cargamos en el registro OPTION.

Ahora cambiamos de banco y habilitamos las interrupciones **GIE**, y en especial **T0IE**, que es la interrupción por desbordamiento del registro TMR0, luego...

```
CLRF   PORTB    ; limpiamos PORTB
```

Lo que viene ahora es preparar la temporización, y de los cálculos que hicimos debíamos cargar 216 en TMR0 y a partir de ahí esperar a que este registro se desborde y produzca la interrupción, entonces hacemos eso justamente...

```
tiempo   MOVLW   0XD8     ; cargo w con 216
         MOVWF   TMR0     ; lo paso a TMR0
```

tiempo es la etiqueta en donde cargaré el registro TMR0 cada vez que quiera hacer una temporización, y 0xD8 es 216 en hexadecimal.

```
NADA  BTFSC   TMR0,7   ; me quedo haciendo nada
      GOTO    NADA     ; hasta que TMR0 desborde, y entonces
      GOTO    tiempo   ; volveré a cargar TMR0
```

La verdad es que ya no tengo nada que hacer, sino esperar a que desborde el TMR0, así es que hice un bucle al cuete, con **BTFSC TMR0,7** estas probando si el Bit7 de TMR0 está a 0, y como ya sabemos que estará a 1, pues ahí te quedas dando vueltas en ese bucle mientras el tiempo pasa, hasta que de repente se produce una interrupción, luego vas, la atiendes y cuando regresas caes en...

```
GOTO    tiempo   ; volveré a cargar TMR0
```

para que comiences a temporizar nuevamente, es decir recargar TMR0 con 216 para luego quedarte en el bucle a esperar la interrupción.

Ahora pasemos a lo mejor de todo esto, **La simulación en MPLAB**, allá vamos...

Simulando interrupciones y temporizaciones con TMR0 en MPLAB

Antes de simular hay que crear un nuevo proyecto, así que eso es lo que haremos, abres MPLAB y si por las dudas te pregunta que si deseas abrir el proyecto anterior seleccionas **No** luego te vas a...

Project --> New project...

y creamos **tmr.pjt** le das a **Ok** y en la ventana **Edit project** seleccionas **tmr[.hex]** y luego a **Node Properties** allí tildas **INHX8M** y **HEX**, confirmamos y nuevamente en **Edit project**, te vas a **Add Node** y escribes **tmr.asm** para ligarlo al proyecto, finalmente le das a **aceptar** y luego **Ok**, bien, ya creamos el proyecto, pero nos falta tmr.asm, así que ve a **File --> New** y ya tienes Untitled1, que mas...???, ah sí... guárdalo como tmr.asm, listo señores, ahora sí...!!!

Vamos al código, lo copias, lo pegas y luego...

Project --> Build All

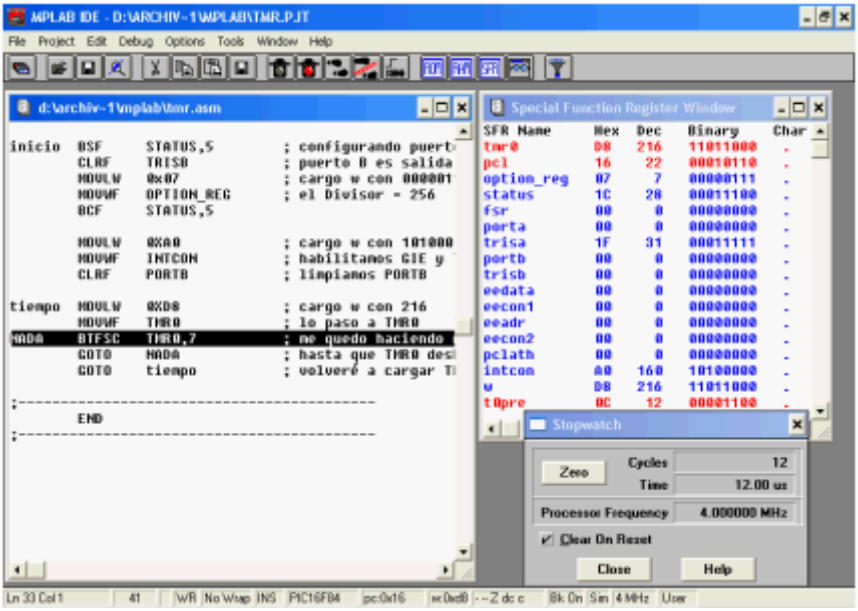
Perfecto...!!!

Abriremos dos ventanas que son para chequear el funcionamiento de la temporización y la interrupción, para ello ve a ...

Windows --> Special Function Register

Windows --> Stopwatch...

Resetea el micro y pulsa el icono “zapatitos” hasta quedar en la siguiente imagen, es decir en la etiqueta **Nada**.


MPLAB IDE - D:\ARCHIV~1\MPLAB\TMR.PJT
File Project Edit Debug Options Tools Window Help
d:\archiv~1\mplab\tmr.asm
inicio BSF STATUS,5 ; configurando puert
CLRF TRISB ; puerto B es salida
MOVLW 0x07 ; cargo w con 000001
MOVWF OPTION_REG ; el Divisor - 256
BCF STATUS,5
MOVLW 0XA0 ; cargo w con 101000
MOVWF INTCON ; habilitamos GIE y
CLRF PORTB ; limpiamos PORTB
tiempo MOVLW 0XD8 ; cargo w con 216
MOVWF TMR0 ; lo paso a TMR0
NADA BTFSC TMR0,7 ; me quedo haciendo
GOTO NADA ; hasta que TMR0 des
GOTO tiempo ; volveré a cargar T
END
Special Function Register Window
SFR Name Hex Dec Binary Char
tmr0 D8 216 11011000
pcl 16 22 00010110
option_reg 07 7 00000111
status 1C 28 00011100
fsr 00 0 00000000
porta 00 0 00000000
trisa 1F 31 00011111
portb 00 0 00000000
trisb 00 0 00000000
eedata 00 0 00000000
eecon1 00 0 00000000
eeadr 00 0 00000000
eecon2 00 0 00000000
pclath 00 0 00000000
intcon A0 160 10100000
w D8 216 11011000
tOpre 0C 12 00001100
Stopwatch
Zero
Cycles 12
Time 12.00 us
Processor Frequency 4.000000 MHz
Clear On Reset
Close
Help

A abrir bien los ojitos que haremos un par de observaciones, primero lo primero...

En Special Function Register

Configuración de puertos, el puerto A no me interesa por que no lo utilizo así que ahí queda, con sus 5 Bit's como entrada (TRISA=00011111), El puerto B está todo como salida (TRISB=00000000), y hemos configurado OPTION_REG como estaba planeado para TMR0 incluso puedes ver el prescaler (los 3 primeros Bit's=111), en el registro INTCON está habilitado **GIE** y **T0IE**, finalmente hemos cargado TMR0 con 216.

En Stopwatch:

Es la primera vez que abrimos esta ventana, y como verás en **Cycles**, tenemos **12**, es decir que hemos ejecutado 12 ciclos de instrucción y estos han consumido 12 microsegundos lo cual puedes ver en **Time**, la frecuencia de procesador utilizada es de 4 Mhz. y está tildado **Clear On Reset**, esto último significa que cuando resetees el micro Stopwatch limpiará todo y lo pondrá a cero, pero como lo que queremos es ver si realmente nuestro programa consume los 10 milisegundos hasta que se desborde TMR0, pues limpiaremos todo a mano, así que dale a **Zero**, y entonces Cycles=0 y Time=0.

Analicemos un poco lo que tiene que ocurrir a partir de ahora, primero que nada, decirte que aquí comienza la temporización de los 10 milisegundo y terminará cuando se produzca la interrupción y salta a ORG 0x04 y como este último es sólo un vector de interrupción pondremos un **Breack Point** en la siguiente línea es decir en...

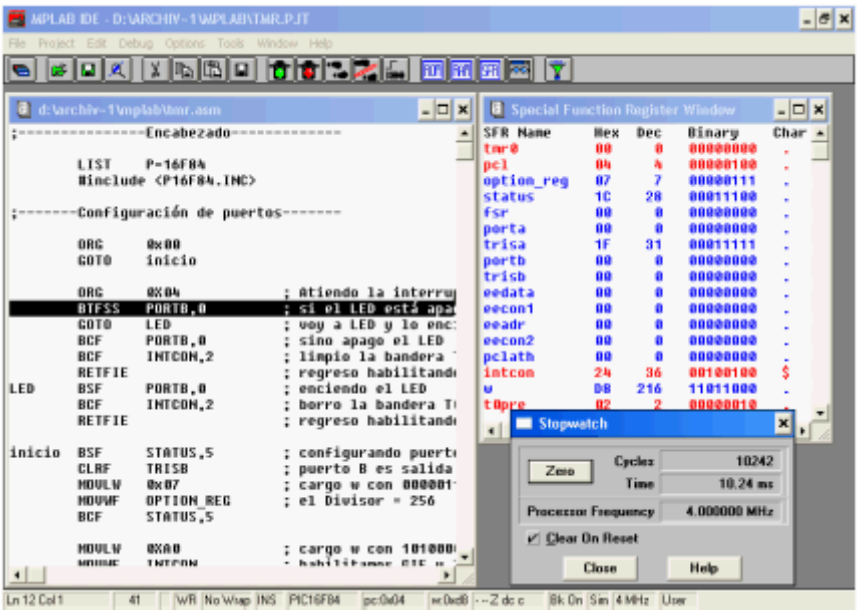
BTFSS PORTB,0

entonces ve a esa dirección y click con el botón derecho, seleccionas **Breack Point(s)** y esta línea se pintará de rojo, lo que hicimos recién es poner un punto de ruptura de tal modo que cuando corramos el programa, comenzará la temporización y cuando se produzca la interrupción, habrán transcurrido los 10 milisegundo y el programa quedará enclavado en **BTFSS PORTB,0**. Si me seguiste y estás bien hasta aquí, daremos el último paso de la simulación...

Vamos... que esto es mas fácil que cebar mate :o))

Haz click en el semáforo verde de la barra de herramientas o bien, ve a **Debug --> Run --> Run**, y que sea lo que Dios diga...!!!

Por suerte se detuvo donde debía, eso significa que estás así...



A observar nuevamente para saber si todo está en su lugar...

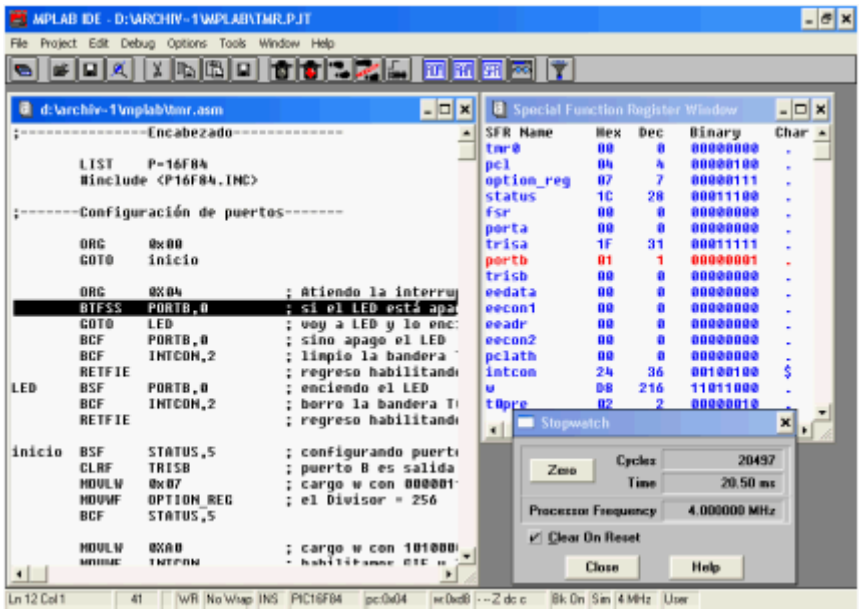
Lo primero que se me ocurre es ver si realmente pasaron los 10 ms

ufaaaaa...!!! me pasé con 242 microsegundos, supongo que debe ser por los decimales, aquellos que aparecieron en los cálculos, pero bueno estamos ahí no crees...???

Sigamos observando... como es lógico el TMR0 se desbordó y está en puros ceros, el registro OPTION_REG no se altera en absoluto y como todavía no encendí el led, PORTB también está en ceros, un último detalle, en el registro INTCON, GIE=0 (interrupciones deshabilitadas), y T0IF=1 (la bandera esta activa)

Te parece si salteamos el Breack Point para dejar que el programa siga corriendo y ver que pasa luego...???

Bien, entonces coloca el cursor donde pusimos el Breack Point y haz click con el botón derecho y luego selecciona **Run To Here** eso hará que el programa continúe, y entonces quedarás nuevamente en el punto de ruptura, así...



Como notarás, pasó lo mismo que hace un momento, sólo que esta vez PORTB tiene encendido el LED o lo que es lo mismo RB0=1 y en Stopwatch, Time=20,50, eso significa que llevamos 2 temporizaciones de 10 ms, yo diría que vamos bien, eso me agrada, si haces **Run To Here** nuevamente, Time se incrementará en 10 ms más y quedará en 30,75 y lo mejor de todo es que el LED se apagará, es decir RB0=0.

Oye...!!!, nos merecemos un aplauso no crees...???, después de tanto despelote con fórmulas, interrupciones y todo eso, estamos listos para temporizar.

En resumidas cuentas... el lío de ecuaciones anterior, si trabajas con un XT de 4 Mhz, se resume en...

Temporización = (255-TMR0) * Divisor de Frecuencia

Esta es la ecuación que utilizaré en todas las temporizaciones ya que siempre trabajo con un cristal de 4 Mhz.

Te imaginas que pasaría si utilizara un registro auxiliar y le cargara el valor 0x64 (100 en decimal) para luego decrementarlo cada vez que se desborda el TMR0...???

Muy sencillo, como el TMR0 se desbordará 100 veces tendrás...

Temporización = 100 * 39 * 256, el resultado es = 998400

Siiii...!!!, prácticamente 1000000 de microsegundos, o sea 1 segundo

Teóricamente esto debería cumplirse, pero por una u otra razón, en la realidad no es tan así, y no queda otra que ajustar las cosas a mano.

OK., hasta aquí llegamos...!!!, por cierto y para aquellos que no se convencen con simple teoría, lean...

Les enseñaré un pequeño truco que cierta vez aprendí observando una lámpara de neón, creo que como profesor de ciencias, ser observador, tiene sus ventajas;

Arma el circuito como siempre lo hacemos, y colócale una resistencia de 220R y un LED en RB0, luego grabas el programa en el PIC, lo montas en el circuito y como dije anteriormente, el LED permanecerá encendido, pero si lo mueves de un lado a otro, verás como parpadea ese LED, debes ser un buen observador, yo lo acabo de hacer, y funciona de diez.

Bueno, que puedo decir, creo que fue suficiente por hoy...

Más formas de Temporizar

Puedes hacer la misma temporización sin utilizar interrupciones, es decir sin habilitarlas, y luego sólo chequear la bandera T0IF, ya que ésta cambiará siempre que se desborde el registro TMR0, y para analizar la bandera nuevamente la vuelves a limpiar, veamos un ejemplo:

Haz el código que quieras y luego llama (Call) a una rutina de retardo, que la puedes etiquetar **retardo**, y cuando la temporización termine le colocas un RETURN, así...

```
            .
            Call    retardo
            .
            .

retardo     BCF     INTCON,2        ; limpias bandera
            MOVLW   0XD8            ; cargas w con 216
            MOVWF   TMR0            ; lo pasas a TMR0

chequear            BTFSS   INTCON,2                ; chequeas T0IF
            GOTO    chequear        ; hasta que TMR0 desborde, y entonces
            RETURN                  ; retornas del retardo
            .
            .
```

En este ejemplo no fue configurado INTCON (para evitar las interrupciones) pero sí, el registro OPTION y del mismo modo que lo hicimos anteriormente, con el mismo prescaler, el resto lo dejo en tus manos, que más...!!!, esto sirve de práctica.

Temporizador sin el registro TMR0

Para seguir con el tema, vamos a tomar de ejemplo una rutinas similar a la que hemos utilizado en la mayoría de los programas que se encuentran en la web y que cumplen la función de generar un retardo cualquiera sin hacer uso del registro TMR0, sino más bien utilizando dos registros alternativos. La idea es averiguar cual es el tiempo que consume esa rutina...

Y aquí la tenemos en vivo...

```
;---------- Inicia Rutina de Retardo ----------

retardo   MOVLW 0x10      ; Aquí se cargan los registros
          MOVWF reg1      ; reg1 y reg2 con 0x10 y 0x20
dos       MOVLW 0x20
          MOVWF reg2

uno       DECFSZ reg2,1   ; Aquí se comienza a decrementar
          GOTO   uno      ; Cuando reg2 llegue a 0
          DECFSZ reg1,1   ; le quitare 1 a reg1
          GOTO   dos      ; iré a cargar reg2 nuevamente
          RETURN                  ; Si reg1 termino de decrementar, regreso del retardo

;------------ Fin Rutina de Retardo ------------
```

Lo que hace este código es lo siguiente; primero carga ambos registros reg1 y reg2, luego comienza a decrementar reg2, cuando éste llega a cero, resta 1 a reg1 y carga nuevamente reg2 y lo decrementa, y cuando llega a cero nuevamente, vuelve a restarle 1 a reg1, en síntesis reg1 se decrementará en 1 cada vez que se vacíe reg2, si te das cuenta reg1 es igual a 0x10 (0x10 = d'16'), por lo tanto reg2 se decrementará 16 veces antes de que reg1 llegue a cero.

Bien, veamos ahora cuanto tiempo consume esta rutina de retardo, y para poder hacerlo deberemos tener en cuenta la cantidad de ciclos de instrucción de cada una de las instrucciones que se encuentran en esta rutina, por lo general la mayoría de las rutinas consumen un ciclo de instrucción salvo cuando realizan un salto, bueno pues en ese caso consume dos ciclos de instrucción. En la siguiente página tienes una tabla con los ciclos de instrucción por si te hace falta.

De las instrucciones que están en rojo; la primera consume un ciclo mientras no se realice el salto y no saltará por lo menos 14 veces (d'14' = 0x20) hasta que reg2 se haga cero, pero la segunda consume dos ciclos y también se repite 14 veces, entonces...

rutina uno = 3 * 14 ciclos = 42 ciclos

De las instrucciones que están en verde; la primera y la segunda instrucción consumen un ciclo y se repiten 16 veces (d'16' = 0x10) es decir reg2 se carga 16 veces antes de que se vacíe reg1, las dos instrucciones restantes consumen 1 y 2 ciclos de instrucción respectivamente y también se repiten 16 veces.

Por tanto tenemos 5 ciclos de instrucción que se repiten 16 veces mas la rutina **uno** que se encuentra incluida dentro de la rutina 2, es decir...

rutina dos y uno = (42 + 5) * 16 = 752 ciclos

Hasta aquí diría que ya estamos, pero el tema es que, cada vez que sales de la rutina que está en rojo e ingresas a la que está en verde, lo haces con un salto es decir 2 ciclos de instrucción y lo haces 16 veces, es decir 32 ciclos más, y por otro lado, desde que la rutina de retardo se inicia, pasaron 4 instrucciones, antes de entrar a la etiqueta uno, eso significa que al total de ciclos de instrucción, hay que agregarle 36 ciclos más, y tendrás en total 788 ciclos.

Si trabajas con un XT de 4 Mhz cada instrucción se ejecuta en un microsegundo, por lo tanto esta rutina de retardo demora **788 microsegundos**

La verdad es que no se cual de las dos temporizaciones es más precisa, si haciendo uso del TMR0, o del modo que acabamos de ver, lo que si creo, es que debes tener en cuenta todos los detalles posibles al hacer los cálculos, tanto en uno como en otro tipo de temporización, lo mejor es evitar los decimales y trabajar en lo posible con números enteros, esto lo digo por las temporizaciones con el registro TMR0, respecto a lo último que acabamos de ver, debes tener mucha precaución en todas las instrucciones utilizadas, para que así no se te escape ninguna.

A demás, con esta última forma de temporizar puedes utilizar el TMR0 para otra cosa, no se... queda a tu criterio, de todos modos ya lo has aprendido y sabes como utilizarlo

Espero haber dado respuesta a muchas de sus dudas, continuaremos aprendiendo todos juntos.

Tabla de Instrucciones y Ciclos de Instrucción:

En las siguientes tablas se pueden ver los ciclos de instrucción consumidos por cada una de las instrucciones del PIC16F84...

Ciclos de Instrucc.	
ADDWF	1
ANDWF	1
CLRF	1
CLRW	1
COMF	1
DECF	1
DECFSZ	1 ó 2
INCF	1
INCFSZ	1 ó 2
IORWF	1
MOVF	1
MOVWF	1
NOP	1
RLF	1
RRF	1
SUBWF	1
SWAPF	1
XORWF	1

Ciclos de Instrucc.	
BCF	1
BSF	1
BTFSC	1 ó 2
BTFSS	1 ó 2

Ciclos de Instrucc.	
ADDLW	1
ANDLW	1
CALL	2
CLRWDT	1
GOTO	2
IORLW	1
MOVLW	1
RETFIE	2
RETLW	2
RETURN	2
SLEEP	1
SUBLW	1
XORLW	1

Es fácil de recordarlas ya que sólo los saltos consumen 2 ciclos de instrucción, observa que los saltos condicionales consumen **1 ó 2**, será 1 siempre que no se realice el salto, ahora si el salto se produce serán 2 los ciclos de instrucción.

Por otro lado las instrucciones de llamada salto y retorno consumen **2 ciclos** debido a que son saltos a una determinada posición de memoria, y bueno, el resto sólo consume uno.

Un Interesante proyecto

Cierta vez, me llamaron de una empresa para reparar un circuito que no funcionaba correctamente, por suerte... me pasaron dos circuitos similares, pero con distinta falla, el tema es que ambos estaban comandados por un microcontrolador muy similar al PIC16F84 (feliz yo...!!!), pero este bicho tenía dos patas demás, y lo peor de todo, es que de fábrica le habían borrado los datos del integrado (a llorar...!!!, pero me las pagarán...!!!), de buenas a primeras, la solución fue sencilla, sólo cambié el micro fallado por el del otro circuito, Arreglé unooooooo...!!!.

Pero esto no se quedaría así..., y decidí hacer uno, pero para el PIC16f84 así es que... a prepararse que les comentaré de que se trata...

El proyecto...!!!

Control para la puerta de un garaje

Al programa le pondré de nombre porton.asm (sin acento), y consiste en lo siguiente...

Se trata de comandar el portón de una cochera (garaje), y con las siguientes funciones. Suponiendo el análisis desde que el portón está cerrado...

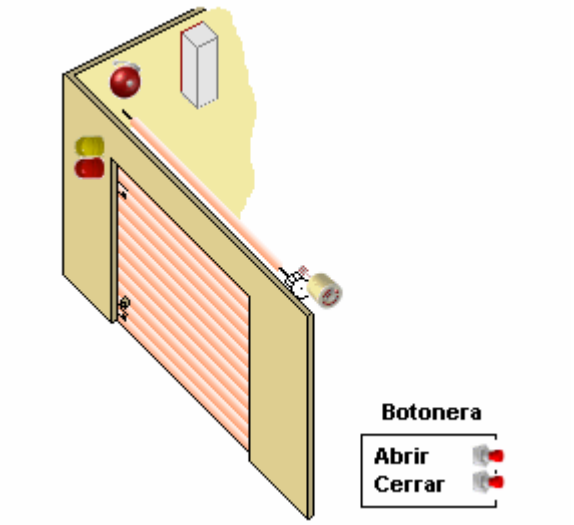
Con un pulsador, llamémosle **"abrir"** el portón comienza a elevarse, al mismo tiempo, se enciende un semáforo pegado en la pared de calle, el cual indica la apertura del portón y que en cualquier momento sacaré o guardaré mi descapotable :o))

Una vez concluida la apertura comienza a gritar un pequeño timbre intermitente.

Con un segundo pulsador, llamémosle **"cerrar"** lo primero que ocurre es que el timbre se apaga, el portón comienza a cerrarse y una vez concluido el cierre, el semáforo se apaga y se enciende la luz del interior de la cochera, la cual permanece encendida por un período de 50 segundos, esto último es debido a que cuando guarde mi Ferrari la cochera quedará a oscuras.

Bueno, pero la vamos a complicar un poquito más, cuando el portón termine de abrirse deberá activar un contacto (pulsador) al cual le llamaremos **Dabrir** (Detiene Abrir), por otro lado, también colocaremos otro contacto para cuando termine de cerrarse y lo llamaremos **Dcerrar** (Detiene Cerrar), estos dos pulsadores vendrían a ser los **fines de carrera** del portón. Finalmente le agregaremos un pulsador de Reset (*ese que siempre ponemos en todos los circuitos con pic*)

Para entender mejor, miren la imagen que muestra la puerta de entrada al garaje:



Creo que ahora está todo más claro...

Antes de empezar a codificar haremos una revisión del funcionamiento de cada uno de los circuitos que se deben incorporar, los cuales serán comandados por el microcontrolador.

Comencemos por el semáforo, éste puede construirse con un 555, y ya lo vimos en el tutorial de electrónica básica, incluso vimos unos cuantos que pueden ser controlados con una señal lógica (del tipo 1-0), esto último lo vimos en el tutorial de electrónica digital (haciendo uso de compuertas lógicas).

Lo siguiente es el timbre, y como no soy muy amigo del área de audio, pues que más, busca por la web, apuesto que conseguirás muchos circuitos, y si tienes uno que sirva para este fin, puedes enviarlo y así completamos este proyecto. Una posibilidad es hacerlo con un 555 pero con una frecuencia que te permita generar sonidos.

Respecto a la luz interna de la cochera, lo más fácil, podemos hacerlo con un relé y listo...

Ahora lo más complejo, como poner en marcha el motor...?

Motores monofásicos de corriente alterna de 4 hilos

El que usé para hacer las pruebas es un motor monofásico obviamente con capacitor y lo más interesante es que posee 4 cables (A, B, C y D), estos se unen de a pares (PAR_{AB} y PAR_{CD}), cuando estos se conectan a la red domiciliaria el motor gira en un sentido, y si intercambias un cable de cada par (suponte PAR_{AC} y PAR_{BD}) obtendrás el giro del motor en sentido opuesto, y ahora me van a matar...!!!, supongo que te estarás preguntando... cual es A, cual B, C y D...???, pues no lo sé, la verdad es que cuando yo lo probé y vi que el motor cambió de giro me puse tan feliz que no busqué más..., de todos modos para mí, esto es a prueba y error...

Por supuesto que si alguien lo quiere saber, que lo comente y lo agregaré en la próxima edición de este manual.

Lo que haremos ahora será analizar un poco lo que deberemos codificar

Analizando Entradas y Salidas

Entradas

Pondremos dos pulsadores ubicados en una botonera a los cuales les llamaremos **Abrir** y **Cerrar**, el nombre indica la función que cumplen así que no diré nada de ellos, luego tenemos los fines de carrera, que no son otra cosa que contactos (sensores) que se activan cuando el portón termina de abrirse (**Dabrir**) o cerrarse (**Dcerrar**). La configuración para estas entradas será la siguiente...

- Abrir <-- RB0/INT (pin6) Pulsador // 0=no_pulsado 1=pulsado
- Cerrar <-- RA0 (pin17) Pulsador // 0=no_pulsado 1=pulsado
- Dabrir <-- RA1 (pin18) Sensor // 0=no_pulsado 1=pulsado
- Dcerrar <-- RA2 (pin1) Sensor // 0=no_pulsado 1=pulsado

Te preguntarás... por qué no estoy utilizando todos los pines del puerto A..???, y es que la mayor parte del tiempo ésta permanecerá con la cochera cerrada, razón por la cual usaremos la instrucción SLEEP para dormir al micro todo el tiempo que no esté en uso, y lo sacaremos de ese estado con la interrupción por el pin RB0 (lo que vimos anteriormente en este mismo tutorial), luego nos queda RA0, RA1 y RA2 que lógicamente son las típicas entradas.

Salidas:

Un detalle a tener en cuenta es que para cambiar el sentido de giro del motor, es preferible hacerlo siempre que éste no se encuentra en movimiento, así que agregué una salida más (Tensión) para quitar la tensión del motor, es más, haré un pequeño retardo (*para evitar desastres...!!!*), Suponte que el motor sube (Tensión=1 y Gmotor=0), para hacer que este baje primero deberás quitar la alimentación del motor (Tensión=0), luego invertir el sentido de giro del motor para que baje (Gmotor=1), esperar un momento y recién entonces ponerlo en marcha (Tensión=1), me explico...???

Ok. ahora sí, uno de los pines (Tensión) activará o desactivará un relé que envía tensión al motor, el segundo utilizará un relé para encender y apagar el Semáforo, el tercero hará lo mismo con el timbre, el cuarto trabajará con dos relés para cambiar el giro del motor (*también podría ser un sólo relé de 8 patas*), la quinta y última salida encenderá la luz de la cochera (*podría ser un relé o un Triac*). La configuración de estos pines será la siguiente...

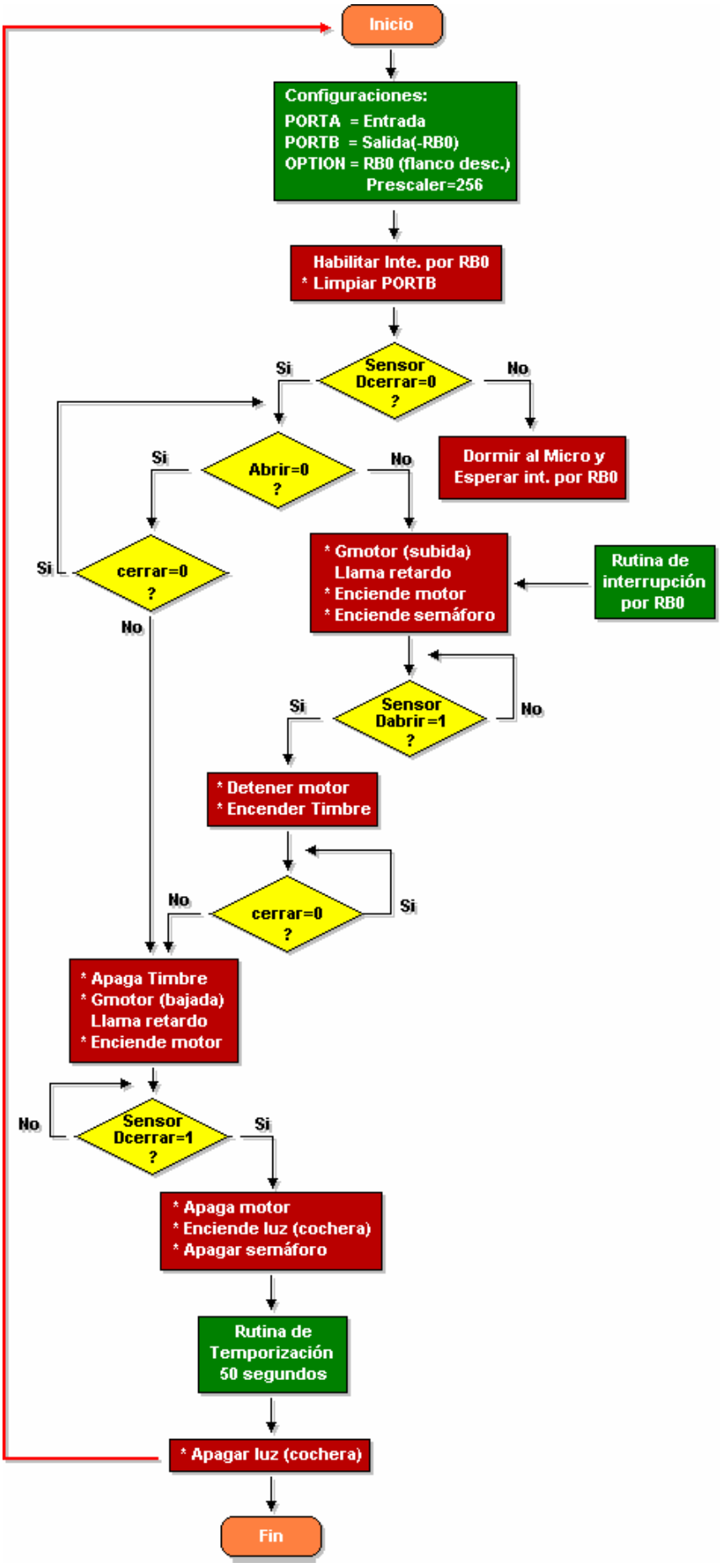
* Tensión	<--	RB1 (pin7)	Alimentación-Motor	//	0=apagado	1=encendido
* Semáforo	<--	RB2 (pin8)	----	//	0=apagado	1=encendido
* Timbre	<--	RB3 (pin9)	----	//	0=apagado	1=encendido
* Gmotor	<--	RB4 (pin10)	Sentido de Giro	//	0=sube	1=baja
* Luz	<--	RB5 (pin11)	Luz-Cochera	//	0=apagado	1=encendido

Ahora haremos un par de observaciones para el buen funcionamiento de la tarjeta de control. Si bien el funcionamiento del pic será secuencial imagínate que pasaría si hoy decido lavar mi móvil, la verdad es que no soportaría el escándalo del timbre ni al semáforo prendiendo y apagando todo el tiempo, pues para eso está el pulsador de reset, por lo tanto cuando sea activado detendré todo el proceso limpiando el puerto B, lo cual haremos justo cuando el programa arranque, es decir en el inicio del código, Entonces... buscaremos el portón, si éste está cerrado, duermo al micro pero si no lo está deberá esperar una instrucción, ya sea abrir o cerrar.

Pero para comprender mejor lo que debemos codificar lo analizaremos con un par de diagramas de flujo (*como hacen los grandes programadores...*)

Diagrama de flujo del código principal

Creo que esta forma de analizar un código simplifica demasiadas explicaciones ya que todo está a la vista. Veamos, lo que está en amarillo es el análisis de pulsadores y sensores, lo que está en rojo y con un asterisco es la activación o desactivación de los pines del puerto B, y lo que no tiene asterisco son simples instrucciones para el programa o llamadas a subrutinas, aquello que está en verde... bueno, la primera es configuración inicial de los puertos, luego una rutina de interrupciones, que una vez atendida regresa el control al flujo del programa. Finalmente la temporización de los 50 segundos, que una vez concluida apaga la luz de la cochera y regresa al inicio del programa para luego hacer dormir al micro con la instrucción SLEEP. Haz click en la siguiente imagen para ampliar el diagrama...


Inicio
Configuraciones:
PORTA = Entrada
PORTB = Salida(-RB0)
OPTION = RB0 (flanco desc.)
Prescaler=256
Habilitar Inte. por RB0
* Limpiar PORTB
Sensor Dcerrar=0 ?
Si
No
Dormir al Micro y Esperar int. por RB0
Abrir=0 ?
Si
No
cerrar=0 ?
Si
No
* Gmotor (subida)
Llama retardo
* Enciende motor
* Enciende semáforo
Rutina de interrupción por RB0
Sensor Dabrir=1 ?
Si
No
* Detener motor
* Encender Timbre
cerrar=0 ?
No
Si
* Apaga Timbre
* Gmotor (bajada)
Llama retardo
* Enciende motor
Sensor Dcerrar=1 ?
No
Si
* Apaga motor
* Enciende luz (cochera)
* Apagar semáforo
Rutina de Temporización 50 segundos
* Apagar luz (cochera)
Fin

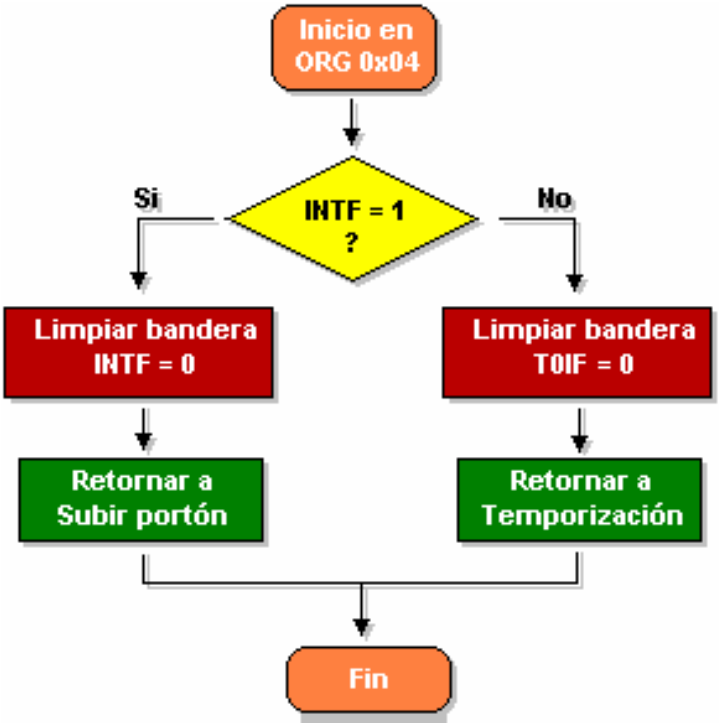
Lo que no se encuentra en este diagrama de flujo es la pequeña rutina de retardo que se utiliza para encender el motor luego de cambiarle el sentido de giro, Tampoco se incluyó la temporización de los 50 segundos porque sino se haría demasiado extenso el diagrama de flujo, pero no te preocupes que lo analizaremos aparte, y nos falta algo más... La rutina de servicio de interrupciones (ISR), que también la analizaremos por separado

Analiza el diagrama de flujo, creo que no requiere más explicación. Continuamos…..

Comenzaré con la rutina de interrupciones, luego la temporización de los 50 segundos y finalmente el pequeño retardo, de acuerdo...???

Diagrama de flujo para el Control de Interrupciones

Como verás la ISR o rutina de servicio de interrupciones es muy pero muy corta, ya que sólo se limita a limpiar las banderas de interrupción y regresar el control al flujo del programa, pero veamos... son dos las fuentes de interrupción, una externa debido a una señal enviada por RB0 que activa la bandera INTF para sacar al micro del modo SLEEP y así iniciar con el flujo del programa, la otra es interna y debido al desbordamiento del registro TMR0 que activa la bandera T0IF, la cual se limpia y luego regresa a la temporización de los 50 segundos (*ya que de allí es que vino*) para cargar TMR0 nuevamente, siempre que el registro Reg1 no haya llegado a cero.



Te preguntarás ahora... con que finalidad utilicé interrupciones?, si lo único que hago es limpiar las banderas cuando estas se producen?

Bueno, no me cuestiones todavía, como dije antes... la mayor parte del tiempo el portón permanecerá cerrado, por lo cual se lo pone en modo de bajo consumo (con la instrucción SLEEP), y se lo saca de ese estado con una interrupción por RB0.

La interrupción por TMR0 es por darle la mayor exactitud posible a los 50 segundos, y a demás por aplicar algo de lo que vimos en este tutorial.

Diagrama de flujo Temporización de 50 segundos

En la figura de la izquierda tiene el diagrama de flujo.

Antes que nada voy a aclarar una cosa... En esta rutina voy a pasar un nuevo valor para el registro INTCON habilitando GIE y T0IE y a la vez deshabilitando la interrupción por RB0, ya que lo único que haré ahora será esperar a que se cumplan los 50 segundos.

Sencillo no...??? bueno, si tuviéramos que hacer los cálculos para obtener los 50... lo haríamos aplicando la ecuación que vimos anteriormente, o sea...

Temporización = (255-TMR0) * Divisor de Frecuencia

Aquí... el divisor de frecuencia es 256 ya que hemos configurado en el registro OPTION los valores **111** para PS2, PS1 y PS0 (el prescaler), por otro lado, la diferencia 255-TMR0 es 39, ya que TMR0 = 216, Teniendo todos estos datos, ya podemos obtener la temporización, la cual vale...

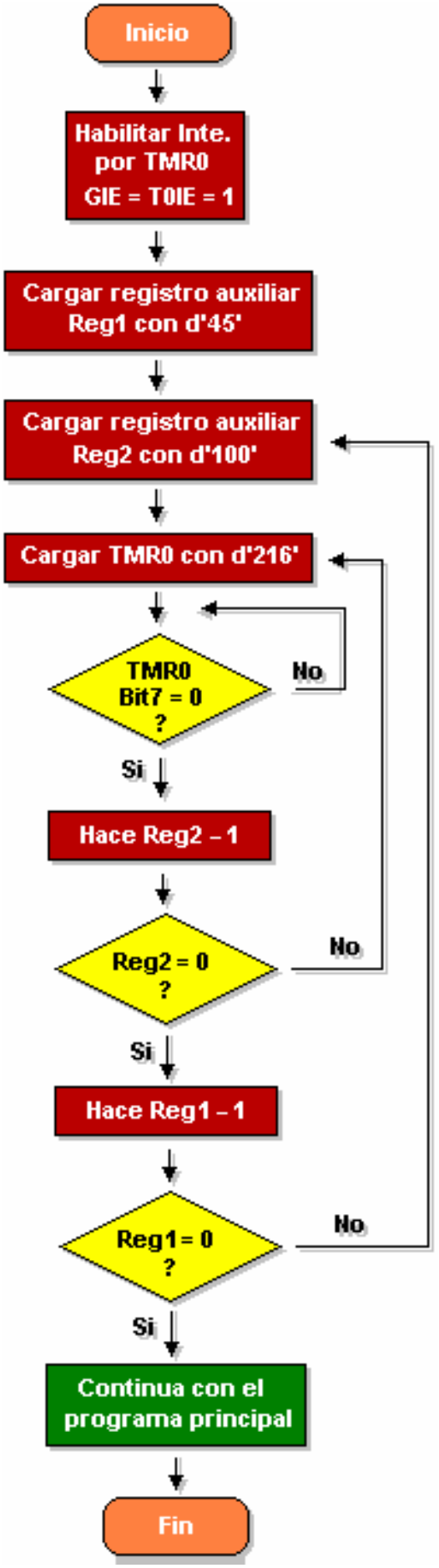
9984 us = 39 * 256

Aproximadamente un milisegundo, pero este **se repite 100 veces** debido a Reg2, razón por la cual obtenemos prácticamente un segundo, y como lo que buscamos es obtener un retardo de 50 segundos pues simplemente usamos un registro más (ya que el anterior no nos alcanza) y **repetimos toda esa rutina 50 veces** (y es lo que se debería cargar en Reg1), te preguntarás porque entonces cargué 45...???, la respuesta es muy simple, la cantidad de saltos en toda esta rutina consumen demasiados ciclos de instrucción y al probar este retardo se me atrasaba casi 7 segundos, pues es eso justamente sólo hice un pequeño ajuste.

Bien, veamos como sería el cálculo si cargaría Reg1 con 50...

Temporización = Reg1 * Reg2 * 39 * 256
49.920.000 useg. = 50 * 100 * 39 * 256

Que bueno no...!!!



La otra rutina de retardo es prácticamente lo mismo sólo que el tiempo que consume es de aproximadamente 189.000 useg. es decir casi la quinta parte de un segundo, y como ya lo analizamos anteriormente pues... no la incluiré...

Ahora viene lo mejor... el código en ensamblador ..!!!

Código para el control de; Portón, Semáforo, Timbre y Luz del garaje (cochera)

Esta vez no haré comentarios respecto al código, ya que habla por sí sólo y a demás ya lo analizamos bastante, no sea cosa que te aburras.

```
;---------------- Encabezado --------------------

            LIST      P=16F84
            #include  <P16F84.INC>

;----------- Registros de tiempo ----------------

reg1        EQU       0x0C
reg2        EQU       0x0D

;-------------- Vector de Reset -----------------

            ORG       0x00
            GOTO      inicio

;----------Rutina de interrupción (ISR)----------

            ORG       0x04
            BTFSS     INTCON,1          ; salta si la interrup. es por RB0
            GOTO      tmr               ; sino, es por TMR0 y ahí lo atiende
            BCF       INTCON,1          ; limpia bandera INTF
            RETFIE                      ; retorno de interrupción
tmr         BCF       INTCON,2          ; limpia bandera de T0IF
            RETFIE                      ; retorno de interrupción

;----------- Configuración de puertos -----------

inicio      BSF       STATUS,RP0
            MOVLW     0XFF              ; carga w con 1111 1111
            MOVWF     TRISA             ; PORTA es entradas
            MOVLW     0x01              ; carga w con 0000 0001
            MOVWF     TRISB             ; RB0=entrada, el resto salida
            MOVLW     0X47              ; carga w con 0100 0111
            MOVWF     OPTION_REG        ; RB0=flanco ascendente, prescaler=256
            BCF       STATUS,RP0

;------- Habilitación interrupción por RB0 -------

            MOVLW     0x90              ; habilitamos interrupciones 1001 0000
            MOVWF     INTCON            ; GIE e INTE(para RB0)
            CLRF      PORTB             ; limpio el puerto B
```

```
;------------ El portón está abierto? ------------

            BTFSC   PORTA,2         ; salta si Dcerrar=0 (portón abierto)
            GOTO    dormir          ; sino (portón cerrado) va a dormir

;------- Si está abierto espera instrucción ------

espera      BTFSC   PORTB,0         ; salta si no se pulsa Abrir
            GOTO    abrir           ; sino, fue pulsado y lo atiende en abrir
            BTFSC   PORTA,0         ; salta si no se pulsa cerrar
            GOTO    c_rrar          ; sino, fue pulsado y lo atiende en c_rrar
            GOTO    espera

;-------- micro en estado de bajo consumo --------

dormir      SLEEP                   ; Espera interrupción por RB0

;--------- Rutina para abrir el portón -----------

abrir       BCF     PORTB,4         ; prepara motor para subir
            CALL    retardo         ; hace un pequeño retardo
            BSF     PORTB,1         ; enciende motor (abre)
            BSF     PORTB,2         ; enciende semáforo

d_abrir     BTFSS   PORTA,1         ; salta si se pulsa Dabrir (sensor)
            GOTO    d_abrir         ; sino lo atiende en d_abrir
            BCF     PORTB,1         ; Dabrir=1, entonces detiene motor
            BSF     PORTB,3         ; y enciende timbre

espero      BTFSC   PORTA,0         ; salta si no se pulsa cerrar (pulsador)
            GOTO    c_rrar          ; sino, fue activado y lo atiende en c_rrar
            GOTO    espero          ; espero que se pulse cerrar

;--------- Rutina para cerrar el portón -----------

c_rrar      BCF     PORTB,3         ; apaga timbre
            BSF     PORTB,4         ; invierte motor para bajar
            CALL    retardo         ; hace un pequeño retardo
            BSF     PORTB,1         ; enciende motor

dcerrar     BTFSS   PORTA,2         ; salta si se activa Dcerrar (sensor)
            GOTO    dcerrar         ; sino, espero que se active
            BCF     PORTB,1         ; Dcerrar=1, entonces detiene motor
            BSF     PORTB,5         ; enciende luz de cochera
            BCF     PORTB,2         ; apaga semáforo

;------- Rutina de temporización 50 segundos -------

            MOVLW   0xA0            ; habilita T0IE interrupción por TMR0
            MOVWF   INTCON

            MOVLW   0x2D            ; cargo w con d'45'=0x2D
            MOVWF   reg1            ; lo pasa a reg1
```

```
tiempo1  MOVLW 0x64          ; cargo w con 100
         MOVWF reg2          ; y lo pasa a reg2

tiempo   MOVLW 0xD8          ; cargo w con 216
         MOVWF TMR0          ; lo pasa a TMR0

nada     BTFSC  TMR0,7       ; Salta si Bit7 de TMR0 es cero
         GOTO   nada         ; sino espera interrupción por TMR0

         DECFSZ reg2,1       ; decremento reg2 y salta si reg2=0
         GOTO   tiempo       ; sino vuelve a cargar TMR0
         DECFSZ reg1,1       ; decrementa reg1 y salta si reg1=0
         GOTO   tiempo1      ; sino vuelve a cargar reg2

         BCF    PORTB,5      ; 50 seg. cumplidos apago luz de cochera
         CLRF   INTCON       ; deshabilito interrupciones
         GOTO   inicio       ; vuelve a empezar para dormir al micro

; ------ retardo de 189000 us. aproximadamente --------

retardo  MOVLW 0xFA          ; Aquí se cargan los registros
         MOVWF reg1          ; carga reg1 con 250
dos      MOVWF reg2          ; y reg2 con 250

uno      DECFSZ reg2,1       ; decrementa y salta si reg2=0
         GOTO   uno
         DECFSZ reg1,1       ; decrementa y salta si reg1=0
         GOTO   dos          ; irá a cargar reg2 nuevamente
         RETURN              ; regreso del retardo

;----------------------------------------
         END
;----------------------------------------
```

Observa aquello que está en rojo...

```
CLRF   INTCON    ; deshabilito interrupciones
GOTO   inicio    ; vuelve a empezar para dormir al micro
```

Eso de deshabilitar las interrupciones lo hice por que ya no será necesaria la interrupción por TMR0 por que terminó la temporización, y lo de regresar al inicio del código es por permitir la habilitación de la interrupción para RB0 y finalmente poner al micro en modo SLEEP.

En la temporización de los 189 milisegundos hay algo muy curioso...

```
retardo  MOVLW 0xFA          ; Aquí se cargan los registros
         MOVWF reg1          ; carga reg1 con 250
dos      MOVWF reg2          ; y reg2 con 250
```

Primero se cargó w con d'250' y luego se pasó el valor a reg1 y a reg2, mientras el decremento en la temporización se ejecuta, hay un salto a la etiqueta **dos**, donde sólo se hace **MOVWF reg2**, Lo curioso es que no se carga nuevamente el registro w con el valor 250 para pasarlo a reg2, y esto

es así por que el registro w no fue utilizado para nada mientras se ejecutaba el retardo, por lo tanto w aún retiene los 250 que le cargamos anteriormente. Cuando hagas esto, debes tener mucho cuidado y asegurarte de que no utilizaste w y que todavía tiene el valor anterior, de lo contrario tendrás muchos problemas...

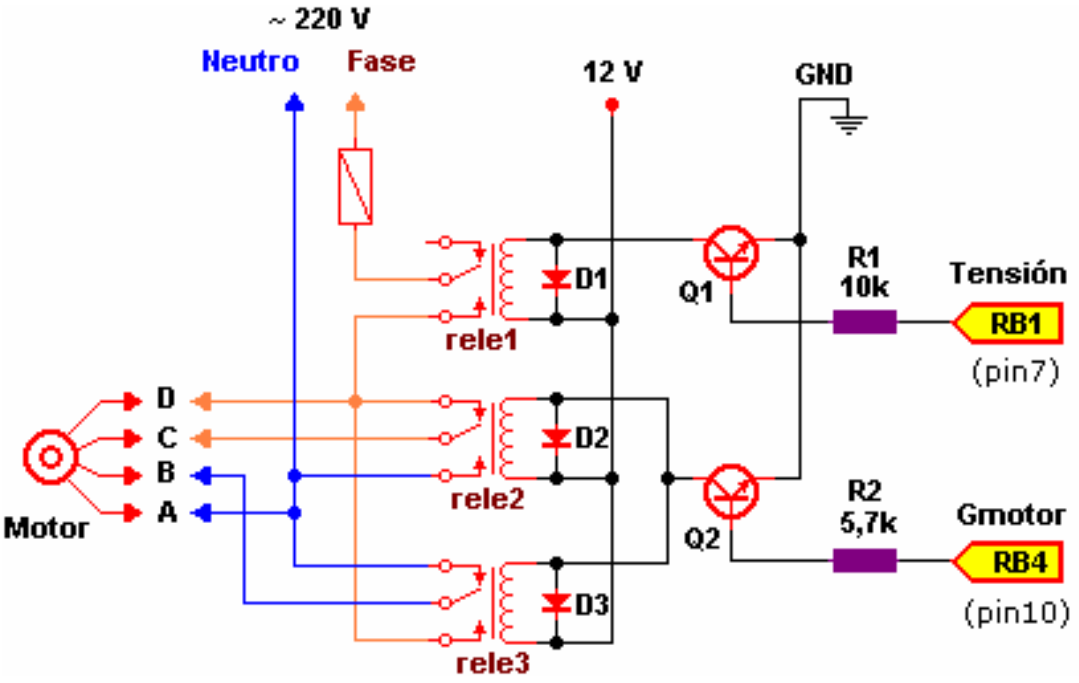
Bueno creo que no queda nada más, o mejor dicho sí, como hacer para conectar los relés, en otras palabras... el circuito...!!!

Hablamos de él a continuación para que no pieras tiempo.

Los esquemas eléctricos

Etapa de Relés para el control del motor

El primer esquema es el relacionado con el encendido y control de giro del motor, en él se incluyen 3 relés, uno para el encendido del motor y los otros dos para invertir los cables de cada par del motor...



Los pares del motor son AB y CD, observa que los cables A y D permanecerán sin cambio mientras que los únicos que cambian son los cables B y C, cuando uno de ellos es **fase** el otro es **neutro**, y viceversa.

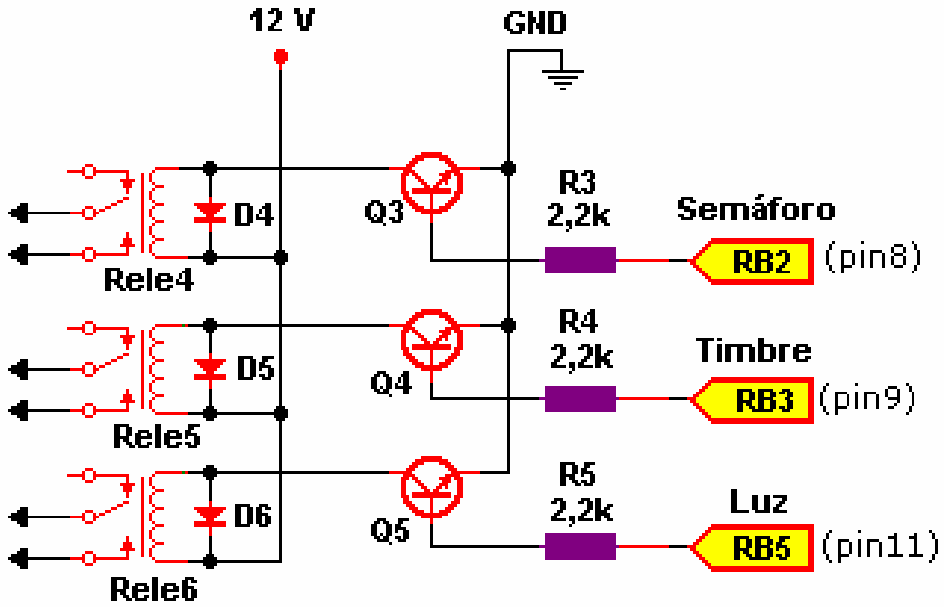
En esta etapa se está trabajando con dos niveles de tensión, la primera es de 220 v y es alterna, justamente lo que requiere el motor para funcionar, la segunda es de 12 v y continua, es lo que requiere el relé para activar sus contactos, a la vez se pueden ver los terminales que deberán ser conectados al micro.

Etapa de Relés para Semáforo, Timbre y Luz

Aquí se encuentran visibles el resto de los relés que controlarán el encendido del Semáforo, Timbre y la Luz interior del garaje...

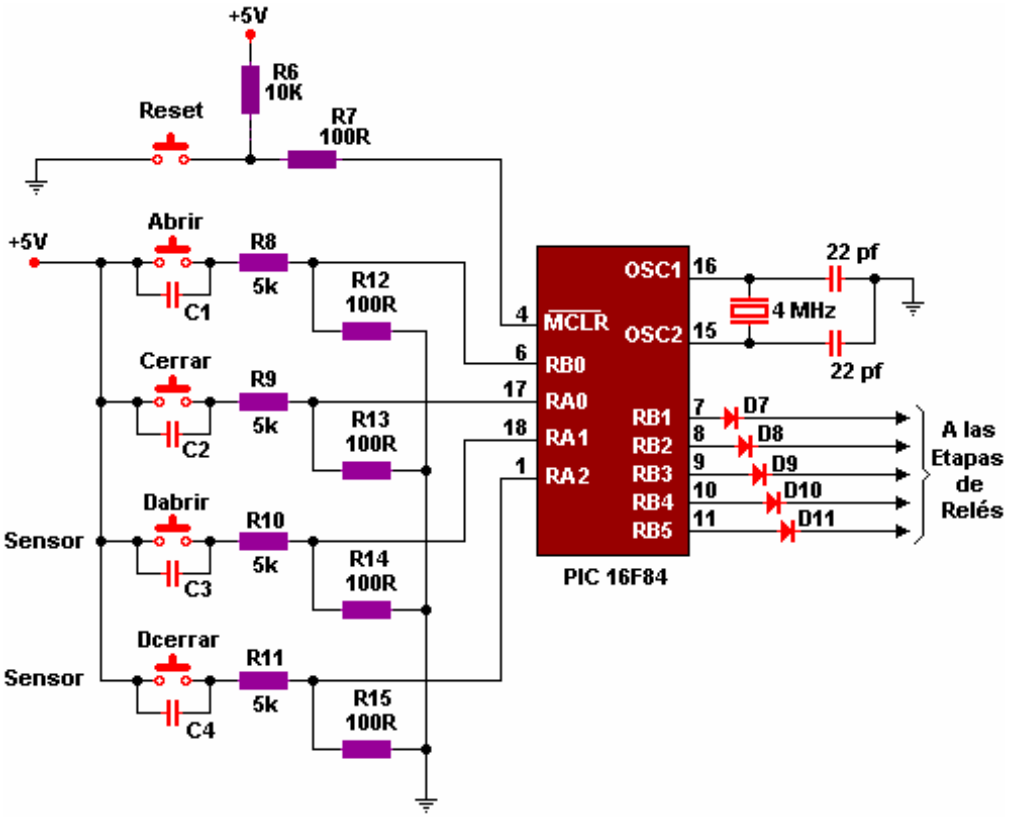
Como podrás notar, en este esquema no se encuentra ni el circuito de Semáforo, ni el del timbre ni la lámpara de la cochera, ya que son circuitos externos a la placa de control, nada quita que lo incluyas si lo deseas.

En ambos esquemas se trabaja con transistores NPN del tipo BC337, por lo tanto la señal enviada a los transistores para que estos se activen debe ser positiva.



Los Diodos son los mismos que utilizamos en las fuentes de alimentación, los 1N4007.

Te comento que estos esquemas fueron sacados de la misma placa que me enviaron para reparación, y si bien en ellos figuran los pines que deberán ir al microcontrolador deberás prever riesgos, y antes de entrar en las etapas de relés colocar un diodo, sería bueno intercalar un Driver entre esta etapa y el micro que bien podría ser el ULN2803 o un Buffer como el CD40106, pero recuerda que estos invierten su señal de salida esto es, si envías un 1 lógico el integrado entrega un 0 lógico en su salida. Yo no incluiré estos integrados en el siguiente esquema pero sí los diodos, así que aclarado esto, veamos como queda el esquema para el micro.



Los capacitores C1 a C4 son de cerámica de 0,1 uf, éstos son para prevenir los rebotes ya que en el código del programa no se hizo ninguna rutina de retardo para evitar rebotes. Los diodos que van a la etapa de relés son del tipo 1N4148, y finalmente habrá que tener en cuenta la fuente de alimentación, que lógicamente debe ser de 5V.

Una cosa más **Dabrir** y **Dcerrar** que figuran como pulsadores en el esquema no son otra cosa que sensores.

Creo no haber olvidado nada, unas palabras finales y damos por concluido este tutorial hasta la próxima actualización...

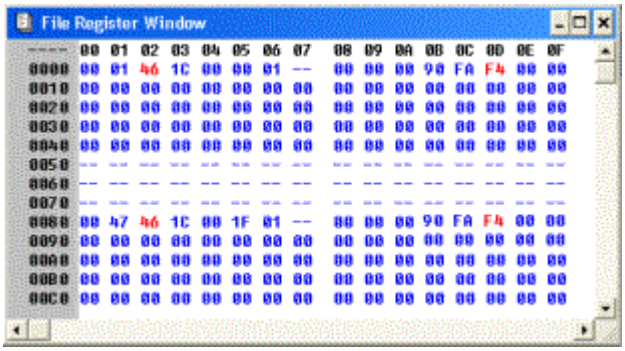
Palabras Finales

Advertí que esto pondría los **pelos de punta...!!!** a muchos, pero si has estado leyendo este manual, no debería traer grandes problemas, ***Palabra mía...!!!***, ya que estás en condiciones de comprenderlo fácilmente, además es sólo una cuestión de interpretación, y de paso vamos incorporando algunas cositas que todavía no habíamos visto, como son los diagramas de flujo, los cuales son muy utilizados al momento de codificar, éstos evitan que te pierdas, al menos a mí me ayuda mucho.

Respecto a la simulación de todo este código con MPLAB, haré algunas aclaraciones, primero que nada, decirte que yo lo hice y con buenos resultados, pero oye...!!! no creas que me quedé a esperar que se cumplan los retardos jajaja... nooooo...!!! ni ahí... lo que hice, fue modificar los valores de los registros reg1 y reg2 y ponerlos a 1 para que la rutina termine rápido y continúe con lo que sigue.

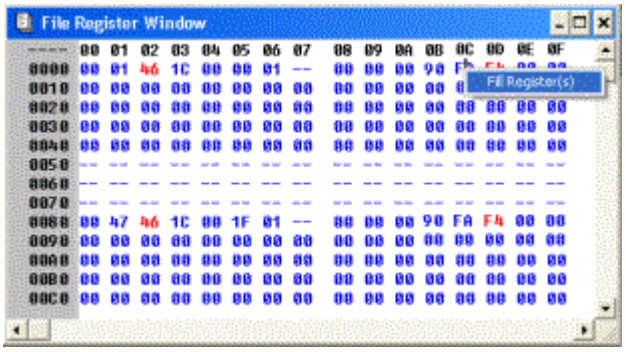
Esto nunca lo hicimos, así que vamos... que te mostraré como hacerlo...

Primero que nada debes tener abierta File Registers Window, que se encuentra en el menú **Window --> File Registers**, y cuando estés corriendo el código y entres a la rutina de retardo lo detienes y verás algo como esto...

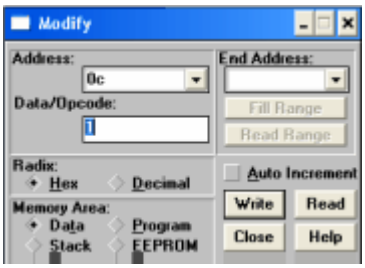


Si recuerdas el código, reg1 es 0C y reg2 es 0D, cuando detuve la ejecución reg2 quedó en **F4**, puesto que se estaba decrementando en la rutina de retardo, y reg1 se mantiene sin cambio y aún retiene el valor que le cargamos inicialmente, es decir **FA**, ahora vamos a modificar los valores de estos registros, así le engañamos a MPLAB para que crea que se decrementó bastante.

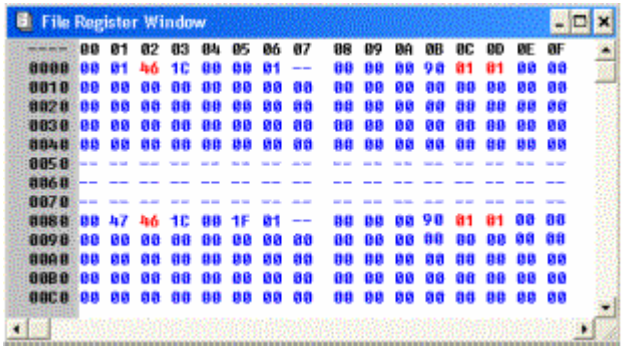
ok, haz un click con el botón derecho del mouse justo en **FA** y luego en **File Register(s)**



y allí te aparecerá un cuadro de diálogo en el que puedes ver justo en **Address** el registro que estás por modificar, en este caso **0c** y en **Data/Opcode** el nuevo valor para ese registro en este caso **1** (este valor es el que estamos ingresando...!!!), tal como se ve en esta imagen...



Luego le das a **Write** (escribir) y habrás modificado el valor del registro **0x0C (reg1)**, para modificar reg2 (0x0D) repites la misma operación y finalmente te quedará así...



Ahora sí, le das a los zapatitos para ver que todo va correctamente y te ahorraste de esperar que llegue el invierno,.

Ni que hablar de la temporización de los 50 segundos, lo que hice allí fue verificar que la interrupción por TMR0 se produzca, que Reg2 decremente cuando esto ocurra y finalmente que Reg1 decremente cuando Reg2 se haga cero, observa que he utilizado los mismos registros en ambas temporizaciones, y no te afecta en nada ya que siempre están a cero si no se está ejecutando alguna de ellas.

Bien, el resto lo verifique grabando el programa en el pic y controlando el tiempo con un cronómetro para saber si realmente se cumplen los 50 segundos.

Por cierto, deberías tener un circuito entrenador si piensas seguir en este tren, ya es hora no crees...???, en la red hay muchos entrenadores de los que puedes disponer, uno más complejo que otro, yo me hice uno que por suerte me sirvió prácticamente para todos los proyectos que hice...

Tambien hay un programa muy bueno para todo esto de los microcontroladores, se llama PROTEUS, es realmente fantástico este programa, pronto hare un manual explicando las funciones más importantes para poder usarlo sin problemas.

Display y tablas

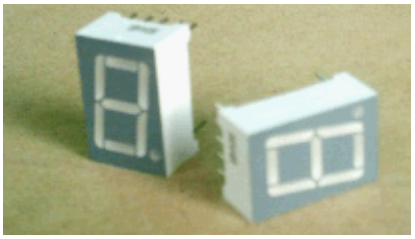
Introducción

Voy a tratar de no hacerlo muy extenso, pensaba hacer algo con LCD, pero lo dejaré para otros manuales más avanzados, vuelvo a repetir con con el programa PROTEUS, simular todos estos proyectos es muy sencillo y no necesitas material algúno. En este caso lo haré con un simple Display, por otro lado, si alguien ya trabajó con los LCD, sería bueno que manden un tutorial al respecto, así lo incorporamos en otras ediciones ó sacaría algo de otro manual que tengo sobre micros, les parece...???. Ok, Comencemos...

Si bien todas las tareas que realizamos, las podemos ver reflejadas en un simple LED, hay ocasiones, en que es necesario ver un número o una letra que nos brinde información de aquello que nuestro micro está haciendo, o simplemente que nos muestre la hora (podríamos hacer un reloj...!!!), o que le muestre un mensaje a nuestra "FIEL" Amada (Te casarías conmigo...???), jaja, bueno, lo último está demás... :oD

Pues para eso están los LCD o los Displays

Para quienes no conozcan lo que es un Display, aquí en el lado derecho tienen una imagen.



Básicamente un Display es una colección de LEDs ubicados de forma estratégica, y como todo LED, obviamente, dispone de un Cátodo y un Ánodo, el tema es que como son varios LED's, se los agrupa uniendo sus cátodos en cuyo caso será de CÁTODO COMUN, o bien agrupando sus ánodos, resultando un Dislplay de ANODO COMUN, por otro lado estos LED's pueden ser fabricados en forma de Puntos o Segmentos, tal es así que se encuentran Displays de 7 segmentos, como los de la imagen (que son los más comunes de todos),

En fin, nosotros trabajaremos con un Display de CÁTODO COMÚN y de 7 segmentos, más el punto (por supuesto)

Formas de Control

Si ya están preparados para comenzar, les contaré lo que haremos...

Vamos a hacer un programa que lea la cantidad de veces que se activa un pulsador y muestre el resultado correspondiente. Para hacerlo, tenemos dos posibilidades, una de ellas es hacerlo en forma directa, es decir conectar el puerto B del micro a los pines del Display, y luego encender cada uno de los segmentos del Display para visualizar el valor correspondiente.

La otra posibilidad es utilizar un decodificador BCD como el 74LS47 o el 74LS249, o el CD4511 que es el que yo utilizaré

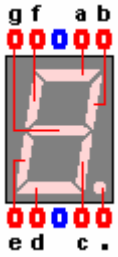
Estos integrados disponen de 4 entradas correspondientes a un código binario, y 7 salidas que se conectan a un Display para mostrar el valor en decimal, o en hexadecimal, según el caso, el nuestro sólo lo hará en decimal.

Yo trabajaré de las dos formas, con y sin decodificador, así tienen una idea de como trabajar con ellos...

Trabajando con un decodificador BCD

Primero veamos todos los componentes que vamos a utilizar

El primero de ellos, es un Display de 7 segmentos de cátodo común, por ser de cátodo común, es obvio pensar que las señales que deberá recibir este Display para iluminar sus segmentos, deben ser positivas, aquí tienen una imagen del display y sus pines.

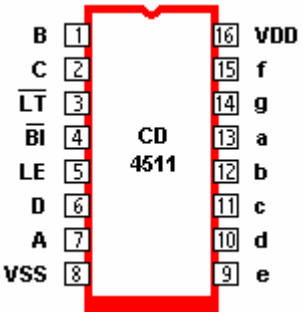


Display

Este Display esta compuesto por 10 pines, de los cuales 7 corresponden al ánodo de cada segmento (nombrados como a, b, c, d, e, f y g), uno para el punto (.),y finalmente 2 que corresponden al cátodo, a los cuales no les puse nombre pero están pintados de azul, aquí hay que aclarar algo, estos dos terminales son comunes, así que da lo mismo que conectes cualquiera de ellos o los dos.

El segundo componente importante aquí es el Decodificador, y yo voy a trabajar con el CD4511 Buscar en este libro las páginas donde estan todas las características de este circuito integrado.

Lo importante de este integrado, es que posee 4 pines de entrada y 7 de salida, mas unos cuantos de configuración. El hecho es que, los 4 pines de entrada (**A, B, C** y **D**) serán los que reciban el código en binario de la cantidad de veces que se activó el pulsador (dato enviado por el micro). Una vez recibido el dato, el integrado se hará cargo de decodificarlo y enviarlo por los pines de salida (**a, b, c, d, e, f** y **g**) para mostrarlo en el display,



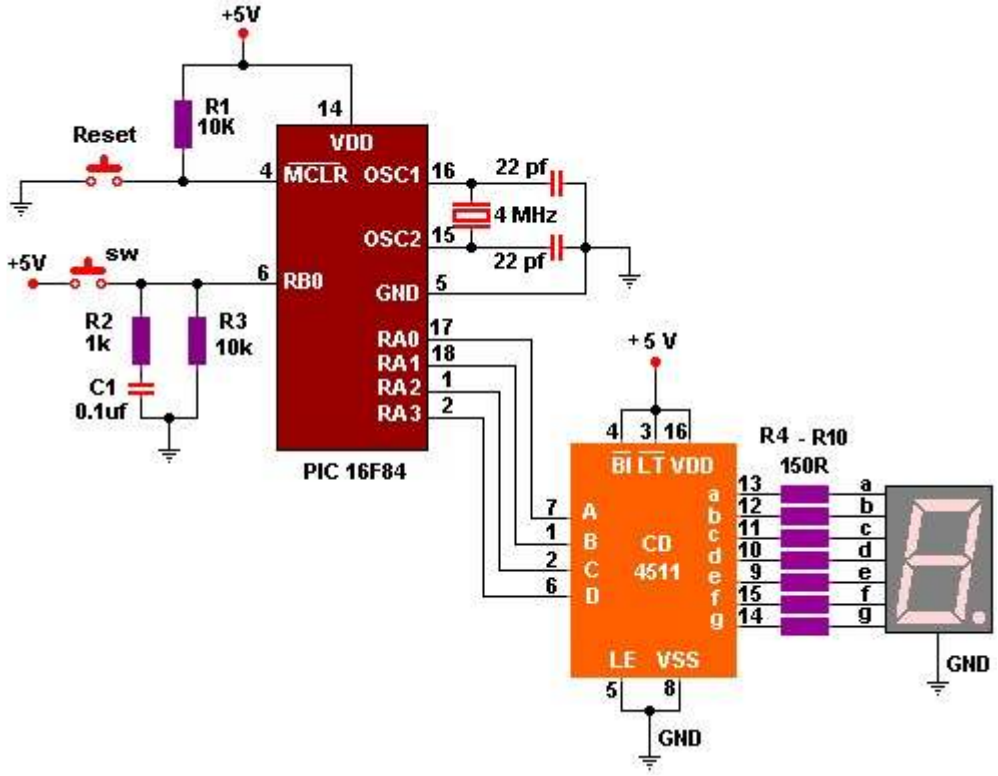
Lo que nos falta saber, es que dato deberé enviar al decodificador para que este muestre por ejemplo un cero, para esto no hay nada mejor que ver su tabla de verdad, y aquí está:

Entradas				Salidas	
LE	BI	LT	D C B A	a b c d e f g	Visualiz.
0	1	1	0 0 0 0	1 1 1 1 1 1 0	0
0	1	1	0 0 0 1	0 1 1 0 0 0 0	1
0	1	1	0 0 1 0	1 1 0 1 1 0 1	2
0	1	1	0 0 1 1	1 1 1 1 0 0 1	3
0	1	1	0 1 0 0	0 1 1 0 0 1 1	4
0	1	1	0 1 0 1	1 0 1 1 0 1 1	5
0	1	1	0 1 1 0	0 0 1 1 1 1 1	6
0	1	1	0 1 1 1	1 1 1 0 0 0 0	7
0	1	1	1 0 0 0	1 1 1 1 1 1 1	8
0	1	1	1 0 0 1	1 1 1 0 0 1 1	9

Por supuesto que de la tabla de verdad, solo tomé lo que me interesa, el resto lo dejé de lado, también se puede notar la configuración de los otros pines del integrado...

El último componente del que hablaremos, es el muy conocido PIC16F84, con el cual nos estamos familiarizando de a poco.

Ahora veamos como es el circuito que vamos a utilizar...



Lo que nos toca ver, es como programar el micro, yo lo haré utilizando la interrupción por el pin RB0, (y así repasamos algo de lo que vimos anteriormente), en él estará conectado el pulsador, y del puerto A usaré los 4 primeros Bits para enviar el dato al decodificador. Ahora bien, si lo que vamos a hacer es un contador, necesitaremos un registro para contar las veces que se activa el pulsador, o bien podemos hacer un incremento directamente en el puerto A, yo lo haré de esta última forma.

Un pequeño detalle antes de pasar a la siguiente página...

Sólo por si las moscas..., si te diste cuenta estamos utilizando 4 bits para enviar el dato al decodificador, y con 4 bits puedes contar hasta 15 (1111), pues resulta que el decodificador solo reconoce los datos hasta el 9 (1001), el tema es que cuando pase a 1010 (10) el display se apagará, ya que será un dato que no reconoce, cosa que deberemos tener en cuenta al programar.

Una solución sería verificar la cuenta, y cuando llegue a nueve reiniciarla en cero, bueno, pero eso lo veremos en la siguiente página...

Bien, ahora presta atención al código que viene.

Código para el Contador

Antes quiero aclarar una cosa, para evitarle problemas a aquellos que no se animan a modificar el archivo P16F84.INC, les muestro una opción, ya que en la red encontrarán otros tutoriales o códigos que utilicen este archivo sin modificaciones.

La idea, es crear una copia de este archivo y renombrarlo, por ejemplo P16F84luis.INC (ponle el nombre que mas te guste...) luego le haces las modificaciones a este archivo.

Bien, ya lo advertí, ahora vamos por el código...

```
;--------------Encabezado------------

            LIST      P=16F84
            #include  <P16F84luis.INC>

;-------Configuración de puertos-------

            ORG       0x00
            GOTO      inicio
            ORG       0x04
            GOTO      ISR
            ORG       0X05

inicio      BSF       STATUS,RP0      ; configurando puertos
            MOVLW     0x10
            MOVWF     TRISA           ; RA0-RA3 = SALIDA
            MOVLW     0xFF            ; PORTB = ENTRADA
            MOVWF     TRISB
            BCF       STATUS,RP0

;-------Habilitación de interrupciones-------

            BSF       INTCON,GIE      ; habilitamos todas las interrupciones
            BSF       INTCON,INTE     ; que sean interrupciones externas

;-------Programa Principal-------

            CLRF      PORTA
espera      SLEEP
            GOTO      espera          ; El micro pasa a bajo consumo

ISR         MOVF      PORTA,W         ; pasamos lo que hay en PORTA a W
            XORLW     B'1001'         ; compara para saber si terminó la cuenta
            BTFSC     STATUS,Z        ; si no terminó salta una linea
            GOTO      reini           ; y si terminó irá a reiniciarla
            INCF      PORTA,F         ; incrementa en 1 PORTA y lo retiene
            BCF       INTCON,INTF     ; borro bandera de interrupción
            RETFIE                    ; regresa al modo SLEEP
reini       CLRF      PORTA
            BCF       INTCON,INTF     ; borro bandera de interrupción
            RETFIE
;---------------------------------------
            END
;---------------------------------------
```

Descripción

Y como siempre, sólo aquello que está en rojo, ya que lo demás lo conocemos desde sus inicios.

#include <P16F84luis.INC>

Respecto a esto no diré nada, ya lo mencioné al comenzar esta sección, vamos por lo otro.

Al configurar TRISA con 0x10 hemos dejado RA4 como entrada, de tal modo que enviemos lo que enviemos al registro PORTA, RA4 no será alterado.

De TRISB, bueno, si bien utilizaré sólo uno de sus pines, configuré todo el puerto B como entrada.

Luego viene la habilitación de interrupciones, la general (GIE), y la que corresponde al pin RB0 (INTE)

Lo siguiente, es limpiar el PORTA, para empezar la cuenta en cero, asi que...

```
CLRF    PORTA
```

y el Display muestra cero "0".

Luego ponemos al micro en espera de la interrupción, con la instrucción SLEEP

Ahora viene el gran secreto, La ISR o Rutina de Servicio de Interrupciones...

Les recuerdo que nuestro decodificador cuenta sólo hasta 9, es decir que si envían 10 por el puerto A, el Display no mostrará nada, por lo tanto, habrá que reiniciar la cuenta, si el puerto A llega a 9 (B'1001') el próximo pulso deberá enviar cero al display para reiniciar la cuenta.

```
ISR     MOVF    PORTA,W         ; pasamos lo que hay en PORTA a W
        XORLW   B'1001'         ; compara para saber si terminó la cuenta
        BTFSC   STATUS,Z        ; si no terminó salta una linea
        GOTO    reini           ; y si terminó irá a reiniciarla
        INCF    PORTA,F         ; incrementa en 1 PORTA y lo retiene
        BCF     INTCON,INTF     ; borro bandera de interrupción
        RETFIE                  ; regresa al modo SLEEP
```

Cuando se presione el pulsador, se generará una interrupción, eso significa que saldrá del modo **SLEEP** para pasar a este trozo de código.

Teniendo en cuenta lo dicho anteriormente, lo que haremos será pasar lo que hay en PORTA al registro w, y luego compararlo con 1001 (9 en el display). Si aún no llegó a 9 saltamos una línea, incrementamos PORTA (INCF PORTA,F) y lo guardamos en el mismo registro, aquí utilicé F (recuerda que antes lo indicábamos con **0** o con **1**, y como estamos utilizando nuestro P16F84luis.INC, pues la cosa se pone más clara), luego borramos la bandera de interrupción y regresamos al modo sleep.

Muy bien. Supongamos ahora, que la cuenta ya terminó...

En este caso, nuestro Display muestra 9, y PORTA está en **00001001**, si es así, cuando hagamos **xorlw** con **00001001**, por ser el mismo valor, la bandera de cero **Z** del registro STATUS, se pondrá en **1**, pues bien, eso significa que la cuenta terminó, por lo tanto habrá que reiniciarla, asi que hacemos un GOTO a la etiqueta **reini**

```
reini   CLRF    PORTA
        BCF     INTCON,INTF     ; borro bandera de interrupción
        RETFIE
```

Lo único que tenemos que hacer aquí, es poner PORTA a cero, el decodificador lo tomará y pondrá el display en CERO, luego limpiamos la bandera de interrupción y regresamos al modo SLEEP.

Bien, Respecto a lo de limpiar PORTA cuando se inicia el código, lo hice de tal modo que puedas reiniciar la cuenta cuando lo desees, simplemente presionando el pulsador de RESET, personalmente creo que este pulsador debería estar siempre en todos los circuitos, y además es importante tenerlo en cuenta, aunque no lo estuviera.

Bueno..., Este fue el modo sencillo para enviar datos a un Display

Ahora lo vamos a complicar un poquitin más, te animas...???

El Registro PCL

Antes de continuar, veamos como trabaja el micro cuando se encuentra ante una serie de instrucciones.

Please...!!!, abstenerse todos los entendidos en el tema, que esto es para duros como yo...!!! (ya lo advertí...)

Existe un registro, llamado PCL, ubicado en la posición 0x02 en el banco de memoria, tiene mucho que ver con el flujo del programa, puesto que le asigna un número a cada línea de código.

Todo empieza con la primera instrucción, esta tiene una posición indicada con un número en el registro PCL, ok. cuando accede a esa posición, se lee la instrucción, se decodifica, y luego se ejecuta, una vez echo esto, el reloj del micro incrementa al contador de programa (PCL) en un unidad, esto hace que el PCL apunte a la segunda instrucción, ahora se lee esta segunda instrucción, se decodifica y también se ejecuta. Nuevamente, el reloj del sistema incrementa el PCL para que apunte a la tercera instrucción, la decodifique y la ejecute. Este proceso se repite hasta que termina el programa (es decir, cuando encuentra un END).

Se habrá entendido...?

Ahora te lo mostraré con un pequeño gráfico, aquí el PCL está representado por una flecha (repito, es un número que indica la posición de cada línea de código), observa.

```
➥  BSF    STATUS,RP0
   MOVLW  0x01F
   MOVWF  TRISA
   CLRF   TRISB
   BCF    STATUS,RP0
```

Bien, de eso se trata, imagínate que te encuentras en un...

GOTO allá (GOTO, es saltar. **allá**, es la etiqueta de un procedimiento).

Es decir, saltar o ir a la dirección donde se encuentra la etiqueta allá, y continuar desde allí..., es decir que al utilizar esta instrucción estás direccionando la secuencia del programa a otra posición.

Piensa, que si Assembler no nos permitiría utilizar etiquetas, deberíamos decirle la dirección del PCL donde se encuentra ese procedimiento, y vaya Dios a saber que número le corresponde a esa dirección, claro que... en realidad no tienes que preocuparte por ello, para eso están las etiquetas.

Te preguntarás que demonios tiene que ver todo esto con lo que estamos viendo, pues bueno, no desesperes, sólo nos falta una cosa más...

Tablas en Asembler:

Me imagino que sabes lo que es una tabla, bueno, una tabla es algo como esto...

Cont. de Programa	ISNT.	DATO
PCL=11 »	RETLW	11000000
PCL=12 »	RETLW	11100001
PCL=13 »	RETLW	00001111
PCL=14 »	RETLW	00111001

Te preguntarás por el contenido de esta tabla, bueno, hablemos de ello...

En esta tabla, cada línea horizontal, es una línea de código, y la dirección de cada línea, está dada por el valor del PCL (el contador de programa), suponte ahora el siguiente código...

RETLW 00001111

RETLW, es retornar cargando **W** con el **L**iteral **00001111**, el problema es que para llegar a esta instrucción deberías pasar por encima de las dos líneas anteriores. La pregunta es, ¿Como se hace eso...?

Para entenderlo mejor, grafiqué la misma tabla, pero sin las líneas de separación, también incluí el PCL y le puse un número de orden en decimal (cualquiera...), esto es sólo a modo explicativo ok...?, observa...

```
PCL
10 ADDWF   PCL,F
11 RETLW   B'11000000'
12 RETLW   B'11100001'
13 RETLW   B'00001111'
14 RETLW   B'00111001'
15
```

La primera instrucción **ADDWF PCL,F** indica que se le debe sumar al registro PCL, lo que hay en W. Con F, le indicamos que guarde el resultado en el mismo registro PCL, es decir...

PCL = PCL + W

El acceso a la tabla lo haremos a través de W, le cargamos un valor y llamamos a la tabla, justo donde al PCL se le suma el valor de W, préstale mucha atención a la siguiente animación, creo que es más fácil de entender...

```
10 ADDWF   PCL,F
11 RETLW   B'11000000'
12 RETLW   B'11100001'
13 RETLW   B'00001111'
14 RETLW   B'00111001'
15
```

Cuando se ejecute esta instrucción el PCL se incrementará en 1 (PCL=11) Si W=0, la suma hará que PCL=11 y pasará a la siguiente línea

Fíjate que en este ejemplo, los accesos a las líneas 11, 12, 13, 14 y 15, se hacen desde la posición 10, la suma con W indica a que línea debe saltar.

Bien, ahora empiezan las advertencias...

- El registro W es de 8 bits, por lo que el máximo valor será 255, ese será el salto más largo que puedas dar.

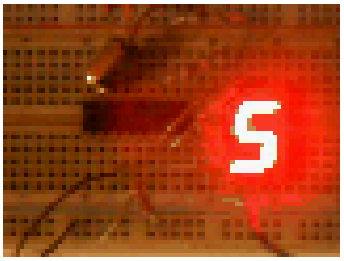
- W no debe superar la cantidad de elementos de la tabla, la del ejemplo anterior tiene 4 elementos por lo tanto el valor máximo de W será 3.
- El acceso a la tabla, se hace sólo para tomar el valor que se busca y regresar al programa principal.
- Los comentarios en una tabla, no son tenidos en cuenta por el PCL, estos son ignorados ...

Bien mis queridos amigos, si lograron comprender bien lo de las tablas, los invito a continuar, que ahora viene lo mejor, aplicaremos todo lo visto en esta sección...

Trabajando Directamente con el Display (Sin Decodificador)

Para no aburrirlos con lo del pulsador, haré que el micro envíe unas cuantas señales por su propia cuenta con un pequeño retardo, lo que haremos será una cuenta regresiva de 5 a 0 y luego haremos que escriba LUIS. (con el puntito incluido), que original, no...?

Como esta vez lo haremos sin decodificador, las cosas se verán totalmente distintas, se parecerá más a un secuenciador que a otra cosa...



El efecto que busco conseguir es este...

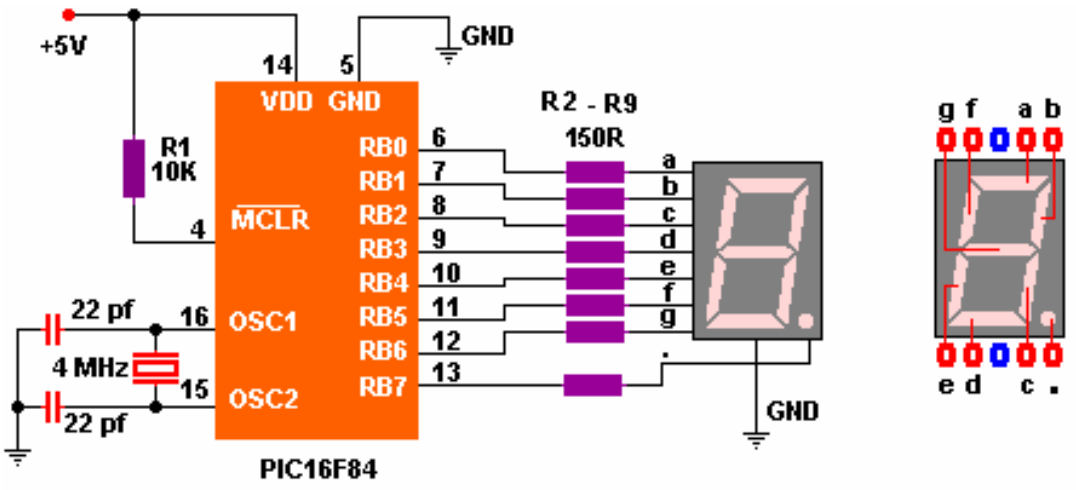
Bien, comencemos...

Esta vez, el decodificador, deberemos crearlo nosotros, por medio de código, y el encendido de los segmentos del Display, se hará activándolos desde el micro. Para que tengas una idea, cuando el micro se encienda por primera vez, el display deberá encender los 5 segmentos que corresponden al número **5**, y luego comenzar la secuencia.

Primero veamos lo que necesitamos...

De componentes, sólo el Display de cátodo común, unas cuantas resistencias de 150 ohm y el micro, ya que todo se hará por programa. Ahora pensemos un poco en los pines del micro que utilizaremos...

Como no haremos entradas de señal, dejaremos el puerto A libre. Del puerto B, utilizaremos los 7 pines más bajos (RB0 a RB6) para activar los segmentos del display, y RB7 para el punto. Bien, eso será para la configuración de los pines del micro, ahora veamos el esquema del circuito...



Nuevamente incluí la asignación de las letras a cada segmento, para que no te pierdas.

A continuación viene lo mejor, "El programa"...

Como haremos una secuencia de caracteres (letras y números) Necesitamos una rutina de retardo que me permita visualizar esa información, también nos hace falta un contador para saber que caracter se mostró en el display y cual es el que sigue, de hecho, a cada caracter le corresponde un código, adivina donde se encuentra ese código...?

siiiiiiiii, en una tabla, esta tabla debe contener el código para los números; 5, 4, 3, 2, 1 y 0, más los caracteres L, U, I, S.

Que tal...?

Ya tenemos todo lo que necesitamos para comenzar, así que vamos por el código

Código para el Control del Display sin Decodificador

En el encabezado incluimos nuestro ARCHIVO.INC y como variables incorporamos **reg1, reg2** y **reg3** para el retardo, más la variable **cont** que controlará la cuenta para incrementar el PCL por medio de W.

En la configuración de puertos, habilitamos PORTB como salida, y comenzamos con la programación.

```
;---------------Encabezado-------------

            LIST      P=16F84
            #include  <P16F84luis.INC>

;----------- Variables utilizadas ----------

reg1        equ       0x0D              ; 3 registros para el retardo
reg2        equ       0x0E
reg3        equ       0x0F
cont        equ       0x10

;---------Configuración de puertos----------

            ORG       0x00
            GOTO      inicio
            ORG       0x04
            ORG       0X05
inicio      BSF       STATUS,RP0        ; configurando puertos
            CLRF      TRISB             ; PORTB = SALIDA
            BCF       STATUS,RP0

;------------- Programa Principal ----------

reini       CLRF      cont              ; pone el contador a 0
            MOVF      cont,W            ; pasa el contador a w (índice)
            CALL      tabla             ; llama a la tabla
            MOVWF     PORTB             ; pasa el dato obtenido a PORTB
```

```
            CALL    retardo

disp_       MOVF    cont,W
            XORLW   B'1001'         ; verifica si el contador llegó a 9
            BTFSC   STATUS,Z        ; si no es así salta una línea
            GOTO    reini           ; si llegó a 9 lo atiende en reini
            INCF    cont,F          ; incrementa el contador
            MOVF    cont,W          ; pasa el contador a w (índice)
            CALL    tabla           ; llama a la tabla
            MOVWF   PORTB           ; pasa el dato obtenido en la tabla a PORTB
            CALL    retardo
            GOTO    disp_

;--------------- Tabla --------------------

tabla       ADDWF   PCL,F           ; se incrementa el contador de programa
                                    ;display     . gfedcba
                                    ;segmentos de los leds del display

            RETLW   B'01101101'     ; código para el 5
            RETLW   B'01100110'     ; código para el 4
            RETLW   B'01001111'     ; código para el 3
            RETLW   B'01011011'     ; código para el 2
            RETLW   B'00000110'     ; código para el 1
            RETLW   B'00111111'     ; código para el 0

            RETLW   B'00111000'     ; código para el L
            RETLW   B'00111110'     ; código para el U
            RETLW   B'00000110'     ; código para el I
            RETLW   B'11101101'     ; código para el S.

;----------Rutina de Retardo----------

retardo     movlw   30              ; Aquí se cargan los registros
            movwf   reg1            ; reg1, reg2 y reg3
tres        movlw   20              ; con los valores 30, 20 y 35
            movwf   reg2
dos         movlw   35
            movwf   reg3

uno         decfsz  reg3,1          ; Aquí se comienza a decrementar
            goto    uno
            decfsz  reg2,1
            goto    dos
            decfsz  reg1,1
            goto    tres
            retlw   00              ; regresare del retardo
;---------------------------------------
            END
;---------------------------------------
```

Vamos por el programa principal...

```
reini   CLRF   cont          ; pone el contador a 0
        MOVF   cont,W        ; pasa el contador a w (índice)
        CALL   tabla         ; llama a la tabla
        MOVWF  PORTB         ; pasa el dato obtenido a PORTB
        CALL   retardo
```

En la primer línea, ponemos el contador a cero, en la segunda, lo pasamos al registro **W**, es decir W=00000000 y nos vamos con este valor a la tabla, veamos que ocurrirá allí...

```
tabla   ADDWF  PCL,F         ; se incrementa el contador de programa
                             ;display. gfedcba   segmentos de los leds del display
        RETLW  B'01101101'   ; código para el 5
```

ADDWF PCL,F es sumarle al PCL lo que trae W, y como W=00000000, pues PCL seguirá siendo igual a PCL, y pasará a la siguiente instrucción...

RETLW B'01101101', recuerda que la línea de comentario no es tenida en cuenta. En esta línea, se carga w con **01101101**, y como se trata de una instrucción de retorno, regresa al lugar de donde vino, es decir a...

```
MOVWF   PORTB   ; pasa el dato obtenido a PORTB
```

```
CALL    retardo
```

Aquí se pasa el valor de **W** a **PORTB** y se visualiza **5** en el Display, luego se hace un retardo, y cuando termina...

```
disp_   MOVF   cont,W
        XORLW  B'1001'       ; verifica si el contador llegó a 9
        BTFSC  STATUS,Z      ; si no es así salta una línea
        GOTO   reini         ; si llegó a 9 lo atiende en reini
        INCF   cont,F        ; incrementa el contador
        MOVF   cont,W        ; pasa el contador a w (índice)
        CALL   tabla         ; llama a la tabla
```

Cargamos W con lo que hay en el contador, y luego, lo que nos toca hacer, es averiguar si ya se mostraron todos los valores que figuran en la tabla, para eso utilizamos la instrucción de comparación **XORLW** con 9 en binario (00001001) puesto que son 10 los elementos de la tabla (del elemento 0 al elemento 9), la instrucción XORLW ya la vimos anteriormente, pero sirve recordarla.

Piensa que si el contador está en 1001 (9), ya mostró todos los elementos de la tabla, y la comparación XORLW dará como resultado 00000000 y la bandera de cero (**Z**) del registro STATUS se pondrá en **1**, de lo contrario permanecerá en **0**, ahora viene la pregunta...

```
BTFSC   STATUS,Z
```

Está en cero la bandera Z del registro STATUS...?, si es así, aún faltan elementos por mostrar, entonces salta una línea, y allí...

```
        INCF   cont,F        ; incrementa el contador
        MOVF   cont,W        ; pasa el contador a w (índice)
        CALL   tabla         ; llama a la tabla
```

Y este trozo de código se repetirá hasta que se muestren todos los elementos.

Bien. Suponte ahora, que la cuenta ya terminó, y se mostraron todos los elementos, eso significa que "cont=1001", cuando llegue a la comparación (XORLW) el resultado SÍ dará **00000000**, la bandera **Z** se pondrá en **1**, y cuando llegues a la pregunta...

```
BTFSC   STATUS,Z
```

Está en cero la bandera Z del registro STATUS...?, la respuesta será NO, por lo tanto se mostraron todos los elementos de la tabla, y no se realizará el salto, es decir que pasará a...

```
GOTO    reini
```

y bueno, allí comenzará todo de nuevo...

Bien mis queridos amigos, espero que les haya servido de ayuda este tutorial, yo lo hice con algo sencillo, para que puedan interpretar la forma de trabajar con estos dispositivos. Imagino que mas de uno, tiene proyectos en los cuales puede incorporarlo, o tiene las intenciones de desarrollar uno nuevo con todos estos chiches, que más da, ahora queda en sus manos, por lo pronto yo me iré a jugar al Mythology, jejeje

BYE...!!!

Cuatro Display's

Parece que esto del Mythology no es lo mío, siempre pierdo...!!!, en fin, veamos con que vamos a seguir...

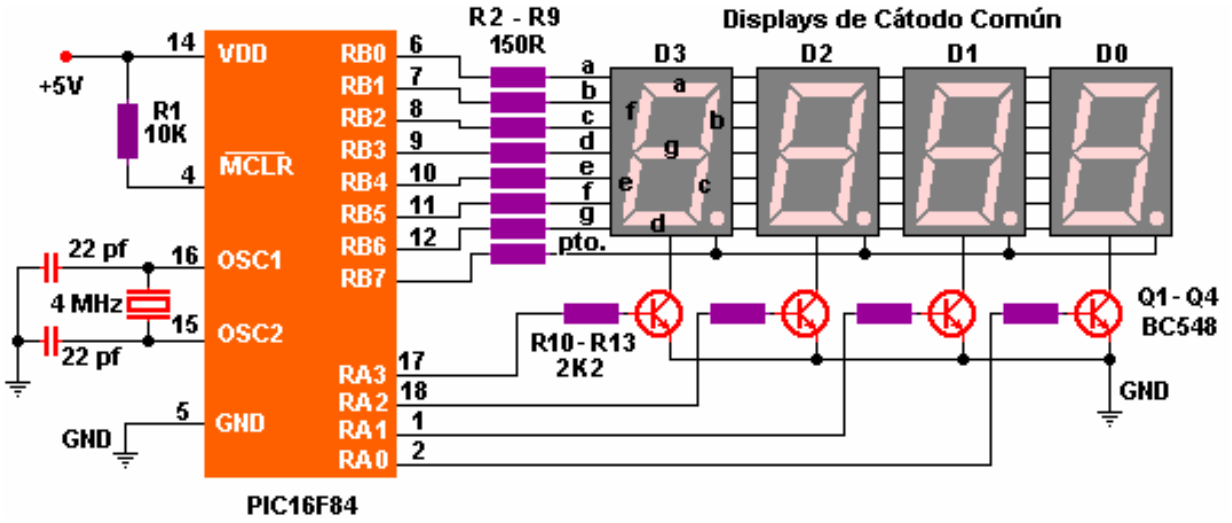
Ah..!!, si..., vieron que interesante fue lo anterior...?, bueno, con las 8 salidas que tiene el micro, nos la arreglamos para manejar un display y activar sus segmentos para mostrar lo que se nos ocurrió, bueno, lo que se me ocurrió.

Imagínate, que pasaría si quisiéramos encender o trabajar con 2 displays, la cosa se complica, a demás no tenemos 16 pines en el micro para los dos displays, y si queremos manejar 4...? uff...!!!, peor todavía...!!!

Bueno, también hay una solución, en este caso la idea es multiplexar las señales enviadas por el micro.

Te preguntarás que es eso de multiplexar, Multiplexar es comooooo, multiplicar, si, es algo así. Algo de esto vimos en el proyecto "secuenciador de 32 canales controlado por PC", claro que allí utilizamos un integrado que se encargaba de mantener la señal enviada por el pc para cada grupo de 8 datos, aquí la cosa será distinta, ya que será el micro quien administre el encendido de cada display y sus segmentos (lo cual se hace por programa).

Para entenderlo mejor, veamos el circuito que vamos a utilizar...



Si prestas atención, el Puerto B se utiliza para enviar los datos a mostrar en cada display, mientras que por el Puerto A seleccionas el display que mostrará ese dato.

Supongamos que quiero mostrar cero "**0**" en cada Display, pues muy fácil, pongo el puerto B en **00111111** (código para el cero), y activo ahora los transistores conectados en el puerto A, haciendo una secuencia de RA0 a RA3, pero sabes cual es el problema...?, que verás correr el cero de un Display a otro, para solucionar este problema, hagamos lo siguiente, realicemos la secuencia tan rápido, que el observador no note el momento en que cambias de display, por lo tanto vería todos los displays mostrando cero, que picardía no...!!! ;))

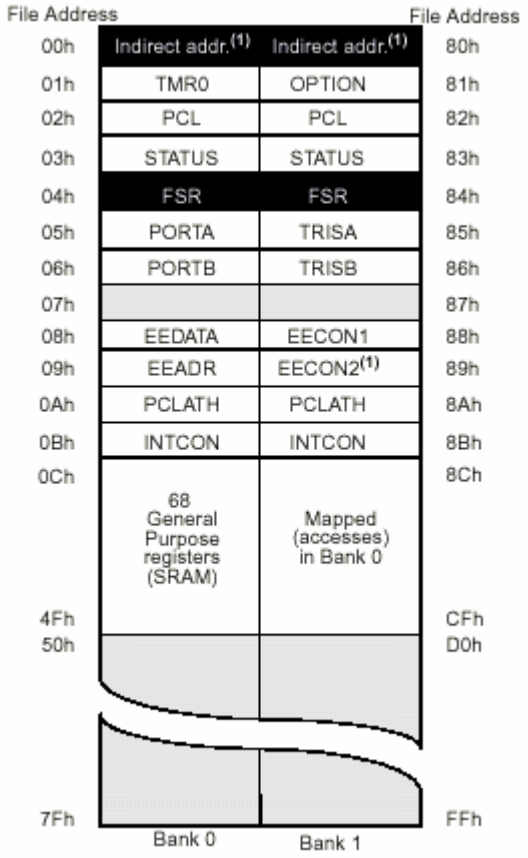
Justamente se trata de eso, ahora, si quisiera mostrar LUIS, enviaría "L", "U", "I" y "S" tan rápido como sea posible, de tal modo que nadie note el cambio de display que estoy haciendo para mostrarlo, algo así...

ejemmmmm..., bueno, el micro lo hará más rápido, y tu verás...

Muy bien, ya está claro lo que haremos, nos falta ver cómo...!!!, para ello vamos a recurrir a un par de registros especiales, de los cuales no hablamos mucho, es más, creo que no hablamos nada de ellos, así que, nos tomamos un tiempo para ver de que se trata...

Direccionamiento Indirecto

Antes de mostrarte los registros de los que hablaremos te traje los bancos de memoria del Micro, en donde los resalté para que puedas notarlo...



Estos 2 registros, y en algunos casos, junto al registro STATUS, pueden trabajar en conjunto para hacer un direccionamiento indirecto de la memoria de Datos (memoria RAM). Bien, que es eso del direccionamiento indirecto...?. Para entenderlo mejor estudiemos estos registros...

Registro 04h (FSR)

Es el Registro selector de registros, es un puntero en realidad, Recuerdas aquello de las interrupciones, pues bien, es la misma dirección, la 0x04h, cuando se producía una interrupción, el contador de programa apuntaba a esta dirección, y nosotros le decíamos por donde continuar, o escribíamos ahí lo que debía hacer.

Ok. Ahora utilizaremos el registro contenido en esta dirección para seleccionar otros registros.

Piensa, que si el FSR es un puntero de registros, pues, en un momento, puede apuntar a uno y en otro momento a otro. Ahora, la dirección del registro al que apunta, se copia en un registro llamado **INDF**, y este último registro, se actualiza en cada cambio del registro FSR, ahora... tienes una idea de lo que es el registro INDF...???

Registro 00h (INDF)

Es el registro para direccionamiento indirecto de datos, a pesar de no ser un registro disponible físicamente (esto lo dice la hoja de datos); utiliza el contenido del registro FSR, para seleccionar indirectamente la memoria de datos o RAM. Si la dirección a la que apunta el FSR se copia en

INDF, una instrucción aplicada a INDF, determinará lo que se debe hacer con el registro al que apunta.

Veamos un ejemplo, de como trabajan estos dos registros, en colaboración el uno con el otro, y así lo entenderás mejor...

Ejemplo de direccionamiento indirecto

- El Registro 05 contiene el valor 10h
- El Registro 06 contiene el valor 0Ah
- Se Carga el valor 05 en el registro FSR (FSR = 05)
- La lectura del registro INDF retornará el valor 10h
- Se Incrementa el valor del registro FSR en 1 (FSR = 06)
- La lectura del registro INDF retornará el valor 0Ah.

Está mas claro verdad...???

Veamos otro ejemplo pero en código. Lo que hace este miniprograma, es borrar el contenido de la memoria RAM entre 0x20-0x2F utilizando direccionamiento indirecto.

```
            ...
            MOVLW  0x20      ; inicializa el puntero
            MOVWF  FSR       ; a la RAM
siguiente   CLRF   INDF      ; borra el registro INDF
            INCF   FSR       ; incrementa el puntero
            BTFSS  FSR,4     ; terminó ?
            GOTO   siguiente ; NO, borra el siguiente
SIGUE       ...              ; SI, continúa con el programa
```

Veamos, Primero cargamos W (W=0x20), luego se lo pasamos al FSR, ahora el FSR apunta al registro 0x20, INDF también.

Borramos el registro INDF (lo ponemos a 00000000), en realidad es el registro 0x20 el que estamos poniendo a 00000000, luego incrementamos en uno el registro FSR, es decir, apunta a 0x21, adivina a quién apunta INDF...?, exactamente..., a 0x21.

Ahora viene la pregunta... El Bit4 de FSR está en uno...?, si es que NO, regresa a **siguiente** y borra INDF (está borrando el contenido de 0x21), ahora incrementa FSR (FSR=0x22=INDF), y vuelve a preguntar, como la respuesta es NO, borra INDF (0x22) y nuevamente incrementa FSR, y bueno, así, hasta que FSR llega a 0x2F, en donde la respuesta a la pregunta es SÍ, y salta una línea para continuar con el flujo del programa.

Viste que bueno que está..., imagínate todas las aplicaciones en que los puedes utilizar, ok. les comento que estos ejemplos fueron extraídos de la hoja de datos del PIC16F84, y creo que están bastante entendibles.

De acuerdo, todo lo que vimos hasta el momento, será lo que aplicaremos para hacer un programa que controle 4 displays.

Listos...???

Vamos por lo que sigue...

Programa de prueba para 4 display's

Volvamos a lo nuestro, y analicemos el programa por partes o en módulos, luego veremos si es necesario un diagrama de flujo...

Primero el encabezado con nuestro archivo .inc para hablar en términos de C, Z, W, F, etc. y la definición de variables...

```
;---------------Encabezado------------------

        LIST    P=16F84
        #include <P16F84luis.INC>

;-------- Variables a utilizar --------

ret1    equ     0x0d
ret2    equ     0x0e            ; registros para retardos
rota    equ     0x0f            ; reg. para rotación (cambio de display)
disp1   equ     0x10            ; primer dato a mostrar
disp2   equ     0x11            ; segundo dato a mostrar
disp3   equ     0x12            ; tercer dato a mostrar
disp4   equ     0x13            ; cuarto dato a mostrar
```

Recuerda que lo que haremos será una secuencia de displays, por lo que es necesario una rutina de retardo, y será muy pequeña, algo como el listado que tenemos en la parte derecha.

```
;------- RETARDO -------

retardo MOVLW 0x03
        MOVWF ret1
dos     MOVLW 0x6E
        MOVWF ret2
uno     NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        DECFSZ ret2,F
        GOTO   uno
        DECFSZ ret1,F
        GOTO   dos
                RETLW  0x00
```

No me voy a gastar explicando el retardo (tema visto anteriormente), sólo lo puse para tenerlo en cuenta, lo que sí rescato de aquí, es el uso de la instrucción **NOP**, que significa no hacer nada (aunque lo que estamos logrando es hacer tiempo). Una cosa más, los registros **reg1** y **reg2** son variables definidas anteriormente.

La configuración de puertos también será sencilla ya que ambos puertos serán de salida uno maneja los datos, y el otro selecciona cada display, entonces...

```
;-------Configuración de puertos-------

reset   ORG     0x00
        GOTO    inicio
        ORG     0x05
```

```
inicio    BSF     STATUS,RP0      ; configurando puertos
          CLRF    TRISA           ; portA es salida
          CLRF    TRISB           ; portB es salida
          BCF     STATUS,RP0
```

Habrás notado que en la definición de variables se incluyeron 4 registros llamados **disp1, disp2, disp3** y **disp4**. Estos registros los vamos a utilizar para guardar el valor que se sumará al PCL en la tabla, de tal modo que tome el dato que queremos enviar al display, y como son 4 displays, pues utilizamos 4 registros y le cargamos con la dirección de esos 4 datos, así...

```
; ------- cargando direcc. de datos de la tabla -------

          MOVLW 0x01
          MOVWF disp1
          MOVLW 0x02
          MOVWF disp2
          MOVLW 0x03
          MOVWF disp3
          MOVLW 0x04
          MOVWF disp4
```

Y ahora la tabla, será muy pequeña, ya que sólo quiero mostrar mi nombre ;o))

```
;------- TABLA -------

tabla     ADDWF  PCL,F            ; se incrementa el contador de programa
          ;display     . gfedcba  segmentos de los leds del display
          NOP
          RETLW  B'00111000'      ; código para la L
          RETLW  B'00111110'      ; código para la U
          RETLW  B'00000110'      ; código para la I
          RETLW  B'01101101'      ; código para la S
```

Aquí también incluí un NOP, para pasar por encima, cuando el programa venga a buscar el primer dato, y así no empezamos desde cero.

Ahora viene lo más importante, el código principal del programa. Primero borramos el Puerto_A para desactivar todos los transistores (apagar los displays) y luego continuamos con el código.

Hay por allí, un registro llamado "**rota**", que lo vamos a utilizar en el siguiente código para activar los transistores que están conectados a PORTA, de tal modo de seleccionar el display que vamos a encender, puesto que son 4, lo vamos a cargar con "00001000" ó 0x08 para seleccionar uno de los displays, y luego lo haremos rotar, para seleccionar los tres restantes. En la siguiente línea, hacemos que el **FSR** apunte al primer registro **disp1**, y nos preparamos para enviar datos al Display, todo esto en las primeras 4 líneas...

```
; ---------- apaga transistores ----------

          CLRF    PORTA

; ---------- PROG. PPAL ----------

ini       MOVLW 0x08
          MOVWF rota              ; rota= '00001000'
```

```
        MOVLW disp1
        MOVWF FSR           ; CARGA FSR CON LA DIRECC. DE disp1

display MOVLW 0x00
        MOVWF PORTB         ; PORTB=00000000

        MOVF  rota,W
        MOVWF PORTA         ; PORTA= 00001000

        MOVF  INDF,W        ; lee dato al que apunta FSR (o sea disp1)
        CALL  tabla         ; llama a la tabla
        MOVWF PORTB         ; pasa el dato al puerto B

        CALL  retardo       ; llama miniretardo
        BTFSC rota,0        ; rota = 00000000 ???
        GOTO  ini           ; si es así, se vio todo, reinicia

        BCF   STATUS,C      ; carry = 0 (para no afectar rotaciones)
        RRF   rota,F        ; rota display
        INCF  FSR,F         ; apunta al siguiente disp_X
        GOTO  display
```

En las dos primeras líneas de la etiqueta **display** enviamos 00000000 a PORTB (puesto que los display's son de cátodo común, los 4 estarán apagados), y luego seleccionamos el transistor del display de la izquierda, esto lo hacemos poniendo 00001000 en PORTA.

Recuerda que el FSR apuntaba a disp1, y como ya sabemos, INDF también, y cuando leamos INDF, estaremos leyendo disp1, luego lo pasamos a **W**, para seguidamente llamar a la tabla, tomar el dato y mostrarlo en el display seleccionado. Como disp1=1 estaremos tomando el código para la letra **L** de la tabla, y lo estaremos enviando al display de la izquierda.

Bien, ahora hacemos un miniretardo, y al regresar, preguntamos si se terminó de rotar, como recién comenzamos..., aún falta..., Ahora bien, por una cuestión de precaución borramos el Carry del registro STATUS, así no se afecta la rotación, de lo contrario cuando terminemos de rotar aparecerán cosas raras como un uno demás, así que lo borramos y hacemos la rotación a la derecha del registro **rota**, luego incrementamos el FSR (para que apunte al registro **disp2**) y regresamos a **display**

Veamos como están las cosas, **rota=00000100**, **FSR=disp2=INDF**, ok, eso significa que ahora, con rota seleccionamos el siguiente display, cuando tomemos el dato de INDF, estaremos tomando el dato de disp2, y de la tabla tomaremos el código para la letra **U**, haremos un retardo, verificamos la rotación y si no terminó, seguiremos rotando, incrementamos el FSR para ir por el siguiente dato, y repetimos el ciclo.

Esta vez **rota=00000010**, **FSR=disp3=INDF**, es decir que esta vez mostraremos la **I**, y seguiremos así hasta mostrar la **S**, cuando esto ocurra, y lleguemos a la pregunta de si terminó la rotación, nos daremos con que SÍ, y entonces saltaremos a **ini**, para repetir la secuencia de displays.

Wowwww...!!!, terminamooooos...!!!, parecía que sería más extenso, pero no, claro que este programita, es con la intención de mostrar usos y aplicaciones del micro, cada uno sabrá la utilidad que le dará, y para que lo pongan a prueba, les dejo el programa completo...

Código para visualizar 4 display's

Recuerda, si quieres ensamblar este programa, deberás modificar el encabezado, cambiando el nombre del archivo **P16F84luis.INC** por el que tu tienes, por lo demás no creo que tengas problemas.

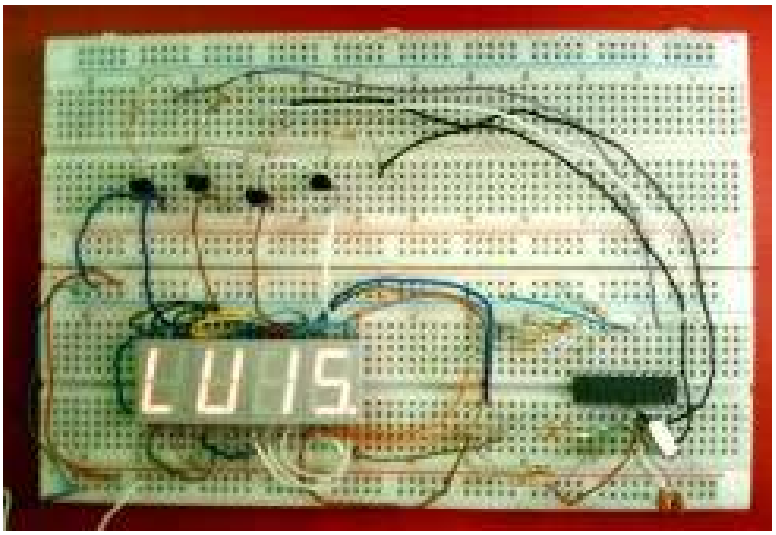
Suerte...!!!

```
;---------------Encabezado------------------

            LIST      P=16F84
            #include <P16F84luis.INC>

;-------- Variables a utilizar --------

ret1        equ       0x0d              ; utilizado en retardos
(milisegundos)
ret2        equ       0x0e              ; utilizado en retardos
rota        equ       0x0f              ; rota el uno para habilitar
displays
disp1       equ       0x10              ; primer dígito a mostrar
disp2       equ       0x11              ; segundo dígito a mostrar
disp3       equ       0x12              ; tercer dígito a mostrar
disp4       equ       0x13              ; cuarto dígito a mostrar

;-------Configuración de puertos-------

reset       ORG       0x00
            GOTO      inicio
            ORG       0x05

inicio      BSF       STATUS,RP0        ; configurando puertos
            CLRF      TRISA             ; portA es salida
            CLRF      TRISB             ; portB es salida
            BCF       STATUS,RP0

; ------- carga de registros a mostrar -------

            MOVLW 0x01
            MOVWF disp1
            MOVLW 0x02
            MOVWF disp2
            MOVLW 0x03
            MOVWF disp3
            MOVLW 0x04
            MOVWF disp4

; ---------- apaga transistores ----------

            CLRF      PORTA

; ---------- PROG. PPAL ----------
```

```
ini         MOVLW  0x08
            MOVWF  rota                ; rota= '00001000'

            MOVLW  disp1
            MOVWF  FSR                 ; CARGA FSR CON LA
DIRECC. DE disp1

display     MOVLW  0x00
            MOVWF  PORTB               ; PORTB=00000000

            MOVF   rota,W
            MOVWF  PORTA               ; PORTA= 00001000

            MOVF   INDF,W              ; lee dato al que apunta FSR (o
sea disp1)
            CALL   tabla               ; llama a la tabla
            MOVWF  PORTB               ; pasa el dato al puerto B

            CALL   retardo             ; llama miniretardo
            BTFSC  rota,0              ; rota = 00000000 ???
            GOTO   ini                 ; si es así, se vio todo, comienza
otra vez

            BCF    STATUS,C            ; carry = 0 (para no afectar
rotaciones)
            RRF    rota,F              ; rota display
            INCF   FSR,F               ; apunta al siguiente disp_X
            GOTO   display

;------- RETARDO -------

retardo     MOVLW  0x03
            MOVWF  ret1
dos         MOVLW  0x6E
            MOVWF  ret2
uno         NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECFSZ ret2,F
            GOTO   uno
            DECFSZ ret1,F
            GOTO   dos
            RETLW  0x00

;------- TABLA -------

tabla       ADDWF  PCL,F               ; se incrementa el contador de
programa
            ;display     . gfedcba     segmentos de los leds del
display
            NOP
```

```
            RETLW   B'00111000'         ; código para la L
            RETLW   B'00111110'         ; código para la U
            RETLW   B'00000110'         ; código para la I
            RETLW   B'11101101'         ; código para la S.
;---------------------------------------
            END
;---------------------------------------
```

Sería bueno verlo funcionar, así que aquí lo tienen...



Es eso simplemente, mostrar un mensaje, y la secuencia entre cada carecter es muy difícil de notar, ya que la velocidad es muy elevada.

Podríamos mejorarlo y hacer que se desplacen los caracteres de un lado a otro, no crees...???, eso lo dejo en tus manos, ya que con todo lo que tienes, puedes hacer lo que se te ocurra, es más, podrías armarte tus display's con LED's comunes, agruparlos en forma de segmentos y trabajar con ellos, que más, bueno, no se, ya verás que es lo que haces, o te quedarás simplemente con esto...???

Proyectos

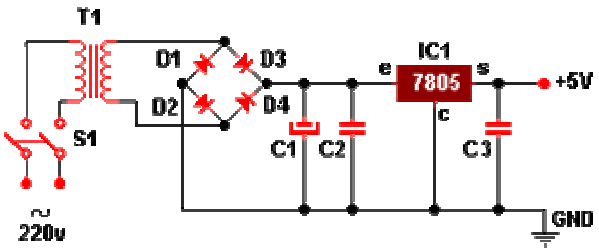
Fuente de Tensión Regulada a 5V

Como las comúnmente llamadas pilas o baterías suelen agotarse en muy corto tiempo es bueno armarse con una de ellas.

¿Qué necesitas para comenzar?...En primer lugar un transformador, que lo puedes extraer de cualquier aparato en desuso, 4 diodos en buenas condiciones, unos cuantos capacitores, y lo que seguramente no encontrarás por allí es un regulador de tensión, estos últimos mantienen un nivel de tensión estable a 5V, 9V, 12V, etc.

Existen los reguladores de tensión positivos y los negativos, se distinguen fácilmente por su nombre. Los primeros corresponden a la serie 78XX y los negativos a la serie 79XX, donde XX es el voltaje a regular. Veamos un ejemplo; si deseas regular la tensión a +5V utilizarás un 78**05**, si deseas hacerlo a +9V acudirás a un 78**09**, y si deseas +12V, bueno... un 78**12**, fácil verdad?...

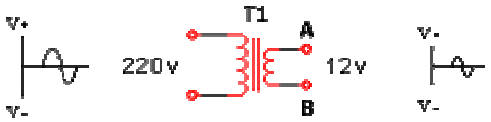
Aquí está el esquema eléctrico de una fuente regulada a +5V...



Para comprender mejor el proceso de rectificación de la fuente, lo vamos a dividir en varias etapas...

Primera Etapa - Reducción de Tensión:

Nuestra red de suministro en Argentina, entrega aproximadamente 220 V de corriente alterna, los cuales reducimos a 12V por ejemplo de la misma tensión a través de un transformador.



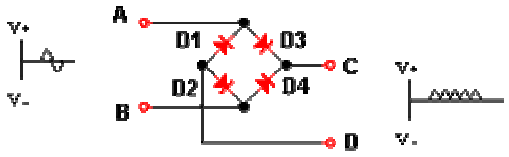
Nota la diferencia de las dimensiones de la onda a la entrada comparada con la salida. Recordemos que la corriente alterna en un momento es positiva y en otro negativa, por lo tanto el nodo **A** en un instante es (+) y **B** (-), en otro instante la polaridad en estos extremos se invierte.

Segunda Etapa - Puente Rectificador:

Vamos a considerar dos momentos;

Primero, con **A (+)** y **B (-)**, únicamente se polarizan el diodo D3 haciendo el nodo **C (+)**, y el diodo D2, haciendo el nodo **D (-)**, D1 pone una barrera y D4 no participa ya que el nodo B es (-).

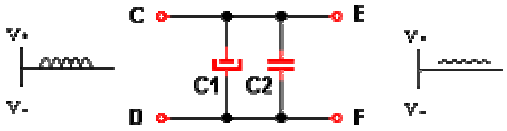
Segundo, con **A (-)** y **B (+)**, únicamente se polarizan D4 haciendo el nodo **C (+)**, y D1, haciendo el nodo **D (-)**, D2 pone una barrera y D3 no participa ya que el nodo A es (-).



Logramos obtener en el nodo C constantes pulsos positivos y en D una constante polaridad negativa, así eliminamos la corriente alterna, o al menos una parte de ella

Tercera Etapa - Los Filtros:

Ok..., acabo de decir pulsos y no necesito corriente pulsante sino continua, o sea que deberemos levantar esos descensos de la curva, y para ello le agregamos dos capacitores

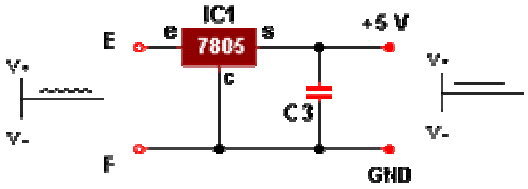


En el momento alto de la curva almacenarán energía para reponerla justo cuando comienza a descender, y entonces en E y F obtendremos la curva que aquí ves.

Aún me queda algo pendiente...

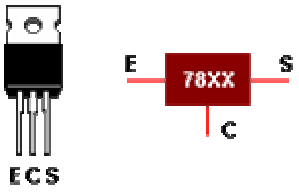
Cuarta y última Etapa - Regulación de tensión:

Fue todo bien pero al momento de consumir energía se regeneran fluctuaciones en la fuente, por esto habrá que compensarlas de nuevo, es aquí donde aparece el regulador de tensión

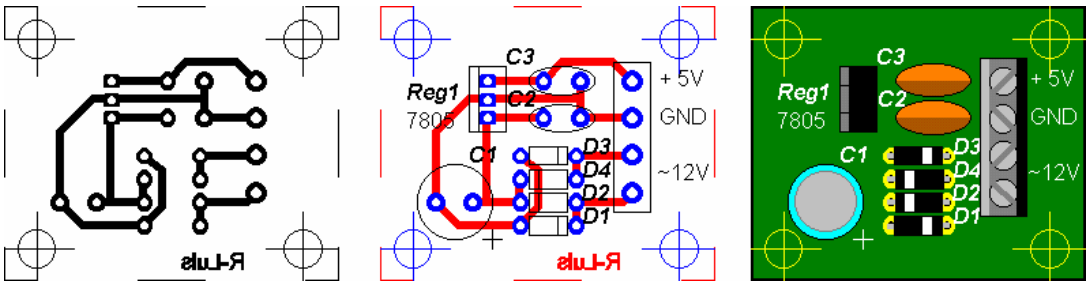


Su tarea será mantener constante el nivel de tensión sin importar el consumo que halla en la salida de la fuente, C3 será su colaborador en esta tarea. Finalmente tendremos un polo positivo **(+5V)** y uno negativo **(GND)**.

Olvidaba dejar la disposición de terminales del regulador de tensión, bueno aquí está, y será la misma para todos los reguladores de tensión positivos, sus terminales son Entrada (**E**), Común (**C**) y Salida (**S**).



Los detalles del pcb para armar esta fuente los puedes ver en la siguiente imagen (diseñado en PCB Wizard)



El proyecto en un sólo zip, Se incluyen en el los siguientes archivos y estan listos para descargar desde la web.

- **fuente.pwz** el pcb diseñado en PCB Wizard (listo para imprimir)
- **fuente.htm** la página que estas leyendo
- **stile1.css** La hoja de estilos
- **imagen** El directorio de imágenes
- **leame.txt** Comentarios

Eso es todo, claro que hay fuentes mucho más simples que esta, pero no viene mal tener conocimiento de ella ya que si decides trabajar con el puerto del PC es mejor asegurarse y diseñar un buen circuito

Más sobre reguladores de tensión estabilizados

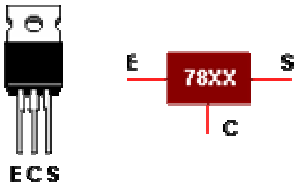
Ejemplo de una fuente estabilizada de 13,8 V.

Con los reguladores de tensión tipo 7805, 7812,7815, etc; podremos diseñar una fuete para nuestras necesidades.

En este ejemplo deberá de tener 13,8 volt, pero como no hay un regulador para esta tensión, deberemos de recurrir a tretas secundarias para esta tensión, mas adelante detallaremos cuales son.

Los Reguladores

En la figura vemos un típico integrado regulador de tensión, a pesar que es parecido a un transistor, estos son circuitos integrados. Tenemos los puntos E, entrada de la fuente de alimentación; T, tierra y S, salida regulada.



El "78", nos indica que es un regulador positivo (existe la serie "79" que regula negativamente); XX nos indica a que voltaje regulara.

Estos pueden ser; 7805, 7806, 7808, 7810, 7812, 7815, 7818, 7822 y 7824. Note que no hay ninguno que regule a 13,8 volt.

El voltaje de alimentación de estos, dependerá del regulador, a continuación les presento una tabla con el voltaje de entrada, mínima y máxima, para que tengan en cuenta.

Tipo de Regulador	Tensión en Volt	
	Mínima	Máxima
7805	7	25
7806	8	25
7808	11	25
7810	13	28
7812	15	30
7815	18	30
7818	21	33
7822	25	36
7824	27	38

Resumiendo, el 7812 nos dará una tensión de 12 volt y el 7815, una tensión de 15 volt. La fuente de alimentación, tendrá que tener unos 3 volt mas, como mínimo, de la tensión de salida.

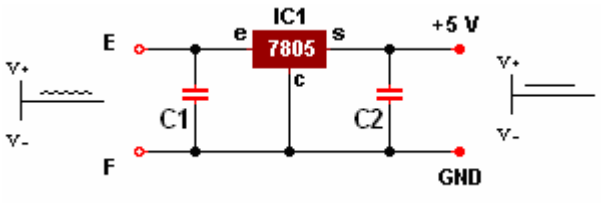
El rango de temperatura será entre 0º y 125º, para una corriente de salida de 1 amper, por lo que es aconsejable colocarle un disipador de calor para evitar que este se queme.

El indice de error en la tensión de salida es de +/- 0,25 volt, así un 7805 podrá tener entre 4,75 y 5,25 volt de salida.

También tenemos que tener en cuenta a la hora de comprar un regulador, el tipo de regulador que compramos ya que algunos vienen para 0,1 amperio y otros hasta 3 amperios. Esto depende de la serie del regulador, estas son las siguientes:

Tipo de Regulador	Potencia en Amperios	Zócalo
78XX	1,0	TO-204, TO-220
78**L**XX	0,1	TO-205. TO-92
78**M**XX	0,5	TO-220
78**T**XX	3,0	TO-204

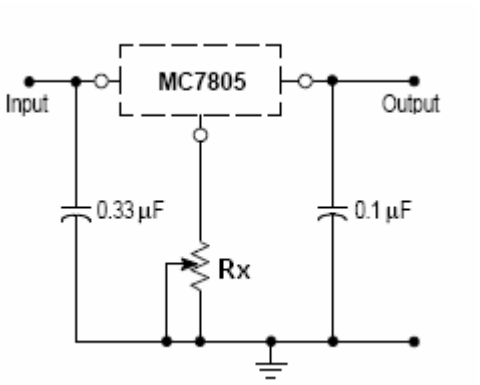
Circuitos



Este es un circuito simple de un regulador de tensión.

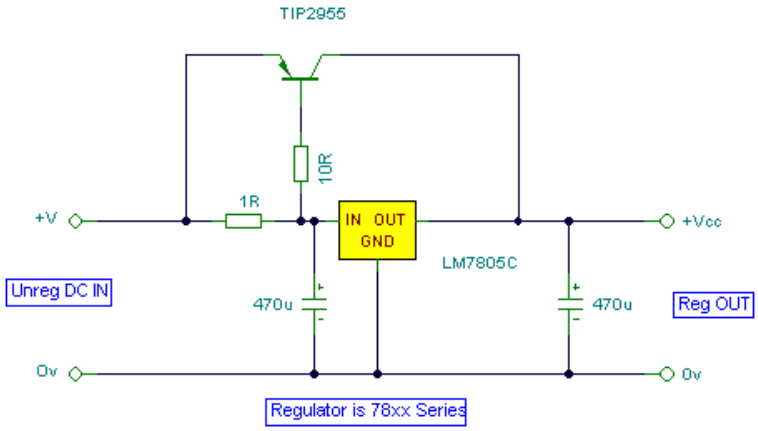
El condensador C1, es un condensador de 0,33 micro faradios, el condensador C2, es de 0,01 micro faradios y forma la carga capacitaba mínima para mantener estable el funcionamiento del regulador. La corriente de polarización del integrado, es prácticamente constante y anda en el entorno de los 4 a 8 mili-ampers

Recuerde que esta estabilización estará limitada a la corriente que soporta el integrado. Si lo que necesitamos es elevar el potencial (voltaje) de salida, deberemos de realizar el siguiente circuito:



Con este circuito más simple que el anterior pero no por eso de menor rendimiento, podremos tener una salida regulada. Tenga presente que la menor salida que tendrá, será la del propio integrado. Supongamos que tenemos un 7805, con una resistencia variable a masa, la menor salida será 5 Volt y la máxima será según la salida del transformador.

Ahora si lo que queremos tener, son corrientes superiores a 1 amper, para ello usaremos el siguiente circuito y que da un resultado espectacular:



Aunque las series 78xx de reguladores de voltaje están disponibles con diversas salidas, usted puede aumentar el amperaje actual disponible con este circuito.

Un transistor de energía se utiliza para proveer extraordinariamente capacidad de carga al regulador, manteniendo el voltaje constante.

Las corrientes de hasta 650mA atravesarán el regulador, arriba de este valor el transistor de energía comenzará a conducir, proporcionando la corriente adicional a la carga.

Este debe estar en un disipador de calor adecuado pues es probable que se caliente.

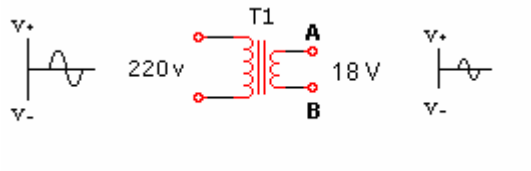
Suponga utilizar un regulador 12v 7812. el voltaje de entrada debe ser algunos voltios más alto teniendo en cuenta caídas de voltaje. Asuma 20 voltios. Asumiendo que la carga sea de 5 amps. La disipación de la energía en el transistor será Vce * ic o (20-12)*8=40watt.

Puede mantenerse caliente el disipador, pero usted necesitará un disipador de calor grande con la buena disipación térmica.

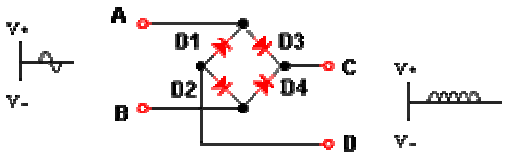
Si usted desea aumentar la corriente de salida con un regulador negativo, tal como 79xx la serie, entonces el circuito es similar, pero un tipo transistor de NPN de energía se utiliza en lugar del PNP.

El Transformador

Para poder regular una tensión a 13,8 volt de salida, necesitaremos una fuente de alimentación mayor que deberá de estar comprendido entre los 18 y los 20 volts dependiendo de la carga.



Podremos usar o bien un transformador de 18 volt en el secundario, o uno de 18 + 18 volt (36 volt con punto medio), la diferencia entre uno y el otro, es que para la rectificación de 18 volt precisaremos un puente de diodos (4 diodos) y con los 36 volt solo dos diodos, uno en cada punta como positivos y el medio seria el negativo.

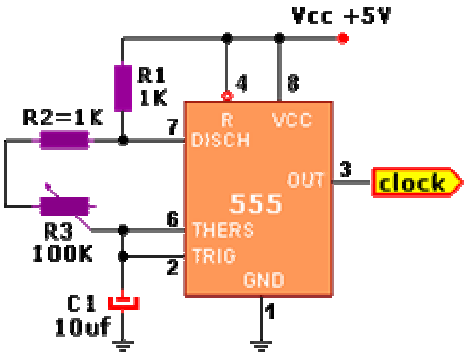


Secuenciador de 5 Canales y 2 Efectos

Consiste en la implementación de un Circuito Integrado CD4017, los datos de este integrado y sus múltiples aplicaciones se encuentran en la sección de ICdatos.

El funcionamiento es muy simple, el integrado CD4017 es un contador divisor por diez con posibilidades de detener la cuenta en un momento determinado y hacer que se reinicie nuevamente, o bien permitirle la cuenta completa, o que se detenga.

Pues eso es justamente lo que haremos, trabajar con estas dos primeras posibilidades...



Lo que necesitamos es un timer para la entrada de reloj de este integrado y con el 555 es suficiente, pero le agregaremos un potenciómetro para regular la velocidad de los pulsos de salida que entrega en el pin 3. Así...

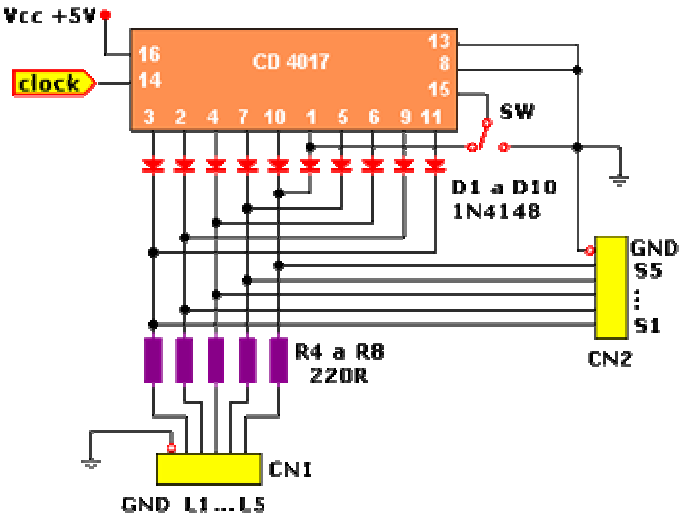
Ahora al CD4017...!!! El pin 13 habilita la cuenta si esta a 0 (GND o VSS) para que esta se reinicie al terminar un ciclo. Si está a 1 (VCC o VDD) realizará la cuenta sólo una vez, y allí quedará, como quiero que continúe la pondré a CERO.

Los pines 8 y 16 son la alimentación del integrado por lo que estarán a GND y VCC (5V) respectivamente, el pin 14 es la entrada de Reloj, allí irá la salida del 555 (pin3), los pines 3, 2, 4, 7, 10, 1, 5, 6, 9 y 11 son las salidas, todas en ese orden, el pin 15...!!! Bueno este será el que me seleccionará el efecto, veamos...

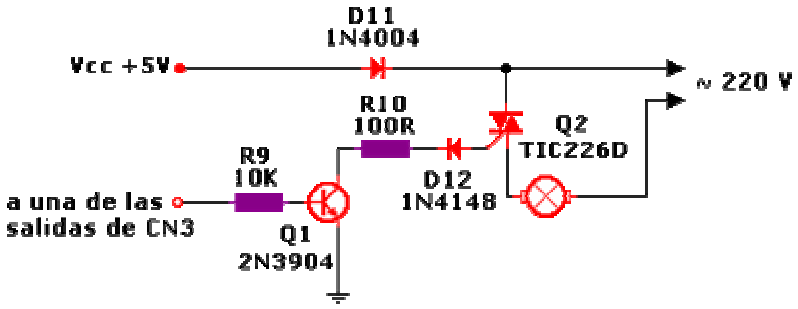
Si está unido al pin 1 la cuenta se realizará hasta 5 y volverá a comenzar (en realidad es hasta 6, pero solo veremos 5), así...

Si está a GND la cuenta será hasta 10, pero observa aquello de los diodos. Cuando cuente 6 se encenderá el 5to LED, cuando cuente 7 se encenderá el 4to, cuando cuente 8 el 3ro y así...

Observa el esquema...

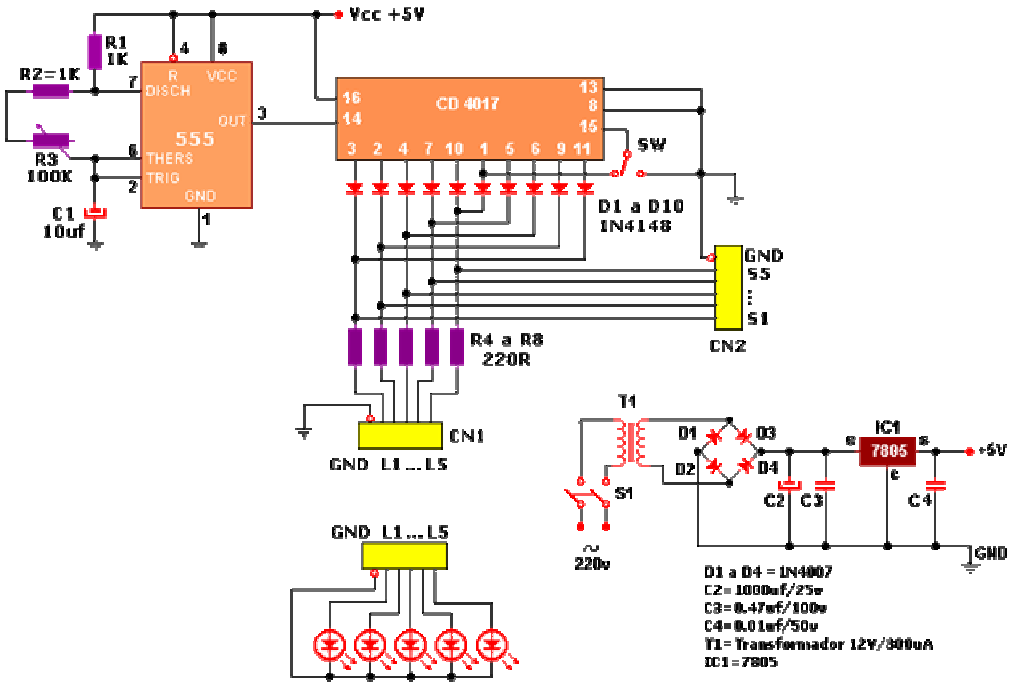


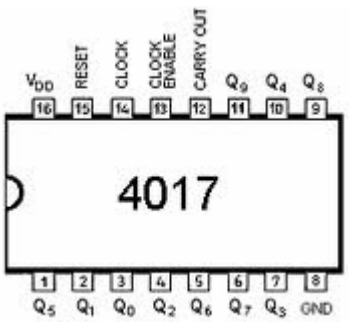
CN1 es el conector que va al monitor de LED's, CN2 es para la etapa de potencia que bien podría ser con transistores o con optoacopladores. El que yo tengo está hecho a transistores y no tuve problemas hasta ahora, el esquema es el que sigue...



Ten en cuenta que sólo representé una de las salidas, por lo que deberás repetirlo para las 4 restantes.

El esquema completo del circuito sería el siguiente...



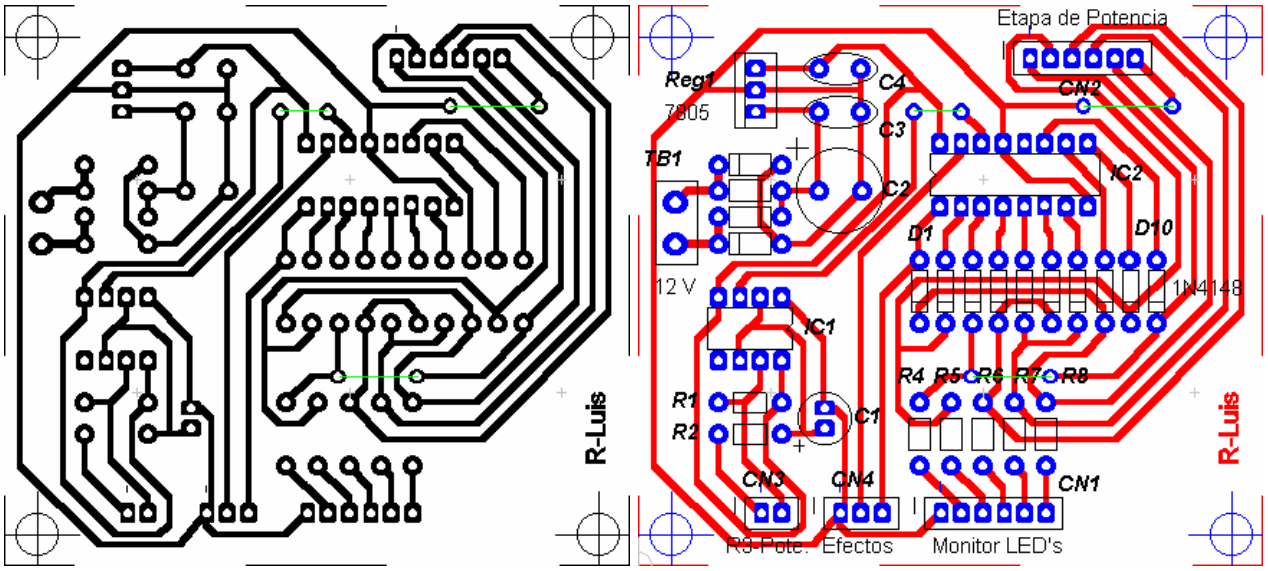


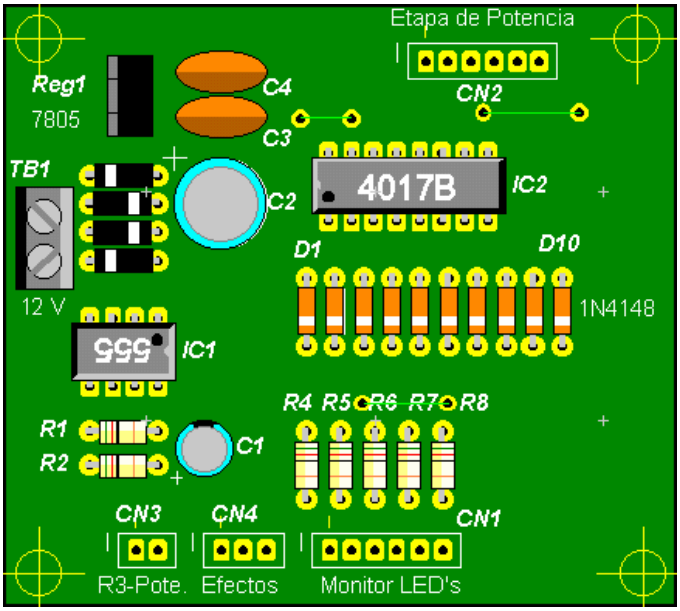
Patillaje del CD 4017

Puedes descargar todo el proyecto desde la Web en el enlace al final de este libro.

Sólo aclarar un par de cosas, el archivo .cyt requiere Crocodile Clips, si no lo tienes ve a la zona de descarga y averigua como conseguirlo. Con PCB Wizard podrías imprimir directamente el pcb (ni que decir, en la zona de descarga...!!!). Para quienes no lo tengan pueden utilizar su programa favorito, de todos modos aquí dejo tres tomas de pantalla del pcb.

Para los más atrevidos, pueden realizar el circuito impreso con su programa habitual.





La verdad, este fue mi primer secuenciador, y aún juego con el...!!!

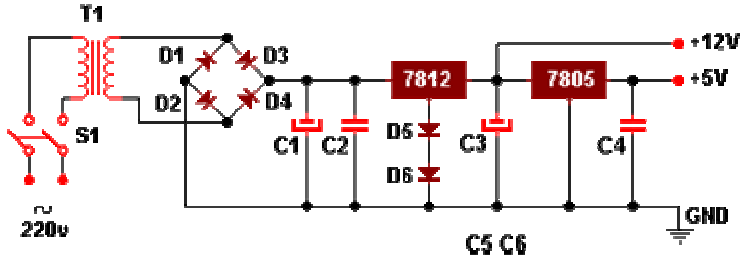
Grabador para PIC's

Ya en el tutorial de microcontroladores mencioné algunos programadores que me llamaron la atención haciendo especial referencia al comentado por José Manuel García Programador PIPO2 y Programador PP2.

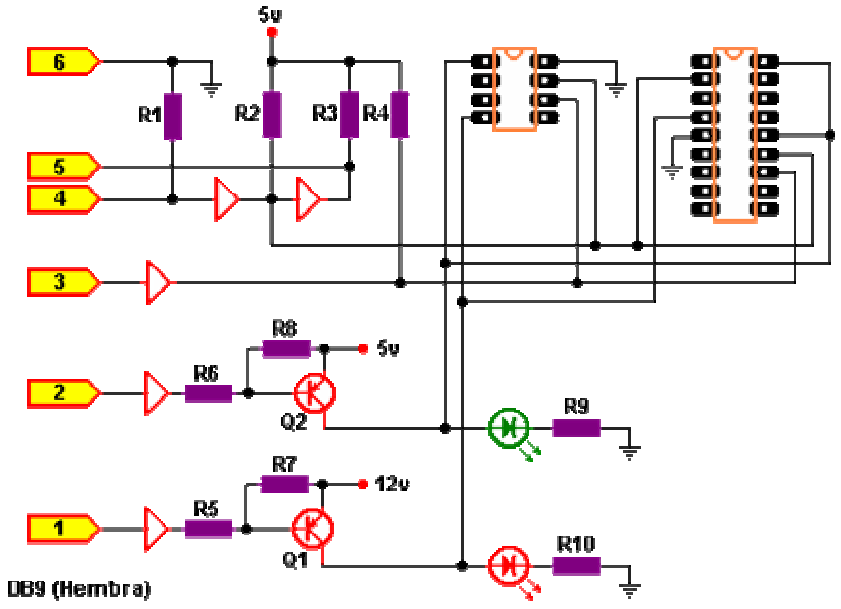
Hay otro que también tiene su buena fama, se trata de TO-20SE, este programador dispone de zócalos para microcontroladores de 28, 18 y 8 pines, es un programador simple y sencillo, y hasta fácil de construir, también se puede encontrar en comercios de electrónica a precios muy competitivos, pero lo nuestro es buscar el esquema y montarlo.

Parece que esto de los programadores no se termina nunca, siempre está saliendo uno nuevo, y como no está demás aquí les dejo el mío.

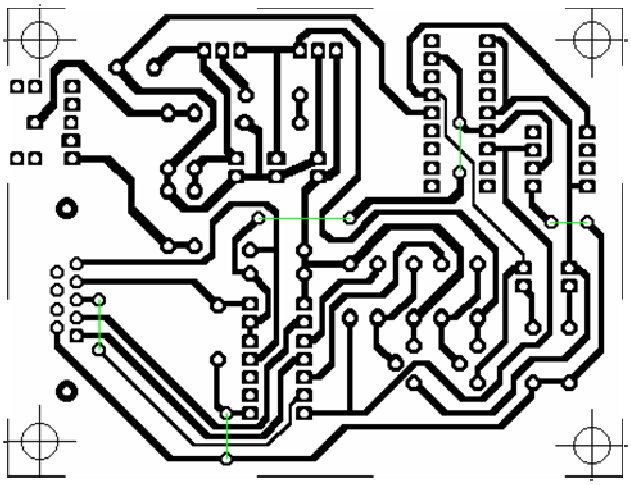
Una breve descripción al respecto, por lo visto muchos tienen problemas con la fuente de alimentación y no alcanzan los niveles de tensión que se requiere para grabar el programa en el pic, mi grabador utiliza esta fuente...

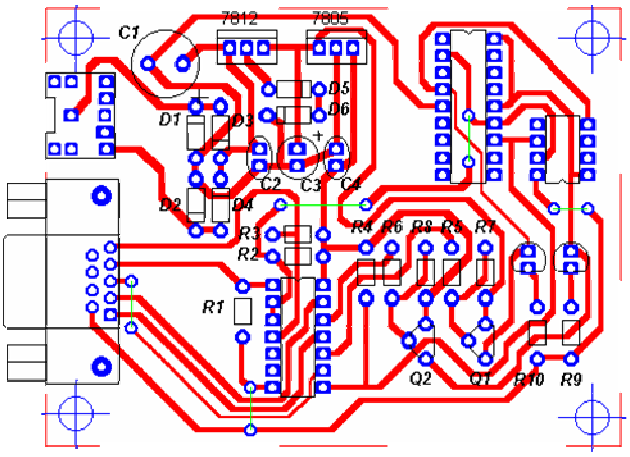


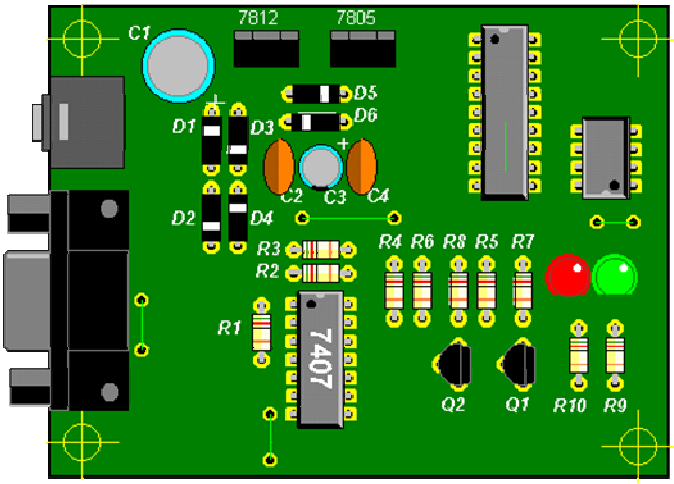
El esquema del grabador es muy simple, sólo debes tener mucho cuidado en no equivocarte al momento de trazar las pistas y montar los componentes, aquí tienes una imagen del esquema...



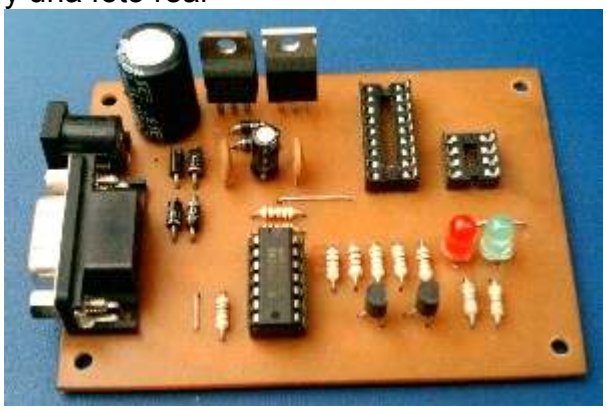
Respecto al pcb, está diseñado en PCB Wizard, listo para imprimir, y sino como siempre digo, puedes utilizar tu programa favorito, aquí tres imágenes del pcb...







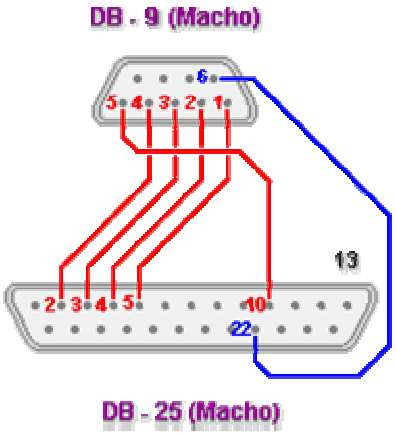
y una foto real



El listado de componentes es el que sigue...

Listado de Componentes	
D1 a D6	1N4007
C1	2200uf/25V
C2	0,47uf/100V
C3	0,1uf/50V
C4	10uf/50V
IC1	LM7812C
IC2	L7805
IC3	SN74LS07
R1, R2, R3, R4, R7, R8	10k
R5, R6, R10	1k
R9	470R
Q1, Q2	BC557
Varios: Conector DB9 Hembra para placa Conector DB9 Macho con tapa Conector DB25 Macho con tapa Transformador de 12 a 15V/0,5A	

El cable para el puerto es de 6 hilos, préstale mucha atención a la conexión de los pines, en la siguiente imagen tienes una vista frontal de los conectores DB9 y DB25, los dos son conectores Macho...



Una cosa más, los microcontroladores soportados según el manual de usuario que vino con mi grabador son los microcontroladores de 18 pines tipo 16C6X, 16C7X, 16C8X, 16F8X, y los PIC's 12C50X de 8 pines.

La forma de uso de este grabador se detalla en este manual en el apartado microcontroladores.

Finalmente, decirte que los daños ocasionados a tu ordenador por el uso de esta información corren bajo tu responsabilidad, por mi parte puedo asegurarte que no tuve inconvenientes y los que monté en base a este grabador tampoco, suerte y recuerda, al trabajar con tu PC debes hacerlo con la mayor precaución posible.

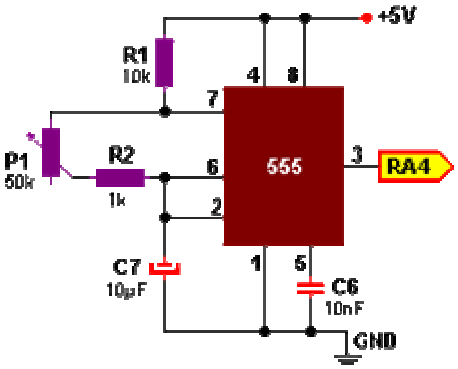
Mi Secuenciador Favorito (8 canales)

Se trata de un Secuenciador controlado con un PIC16f84. Si quieres conocer los detalles para la programación del micro ingresa en la sección de microcontroladores y apunta al tutorial de MPLAB, o bien ingresa por aquí, por supuesto que también allí se incluyen los 16 efectos del secuenciador para que puedas verlo y son algo así.

Como dije anteriormente... La verdad es que no es originalmente mío, de todos modos basta analizar como se manejan las entradas y las salidas del micro, para comprender como se lo debe programar.

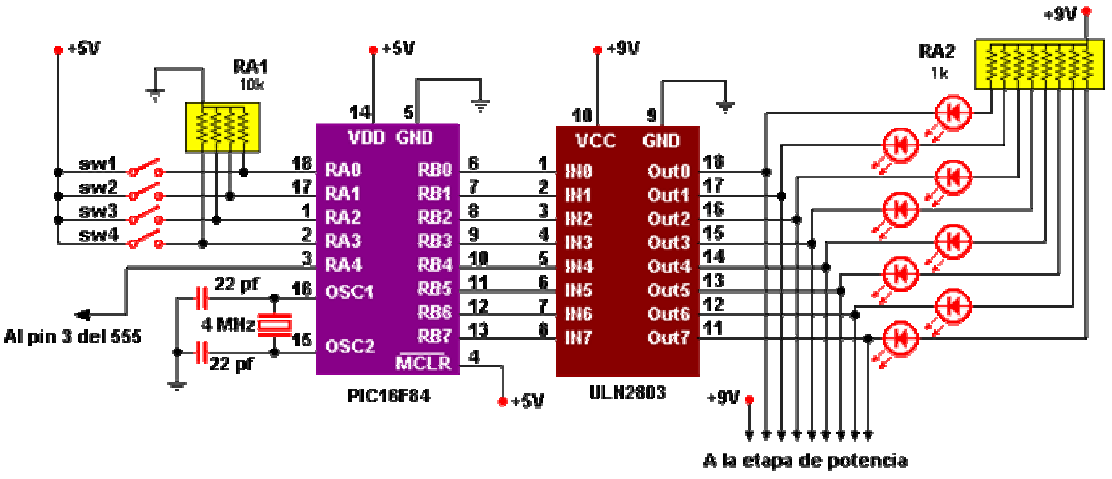
Si ya viste los efectos, te darás cuenta de que se trata..., veamos, este secuenciador tiene 8 salidas 4 interruptores para seleccionar uno de los 16 efectos y una entrada para un timer (en este caso con un 555), es decir que del micro no nos quedó ningún terminal libre.

Entonces vamos por partes, El circuito del 555 debe disponer de un control de velocidad, por lo que le pondremos un potenciómetro, bueno, y si quieres más velocidad cambia el capacitor de 10 uf por uno de 1uf, la salida (el pin 3) la conectaremos al terminal RA4 del micro, observa el esquema.

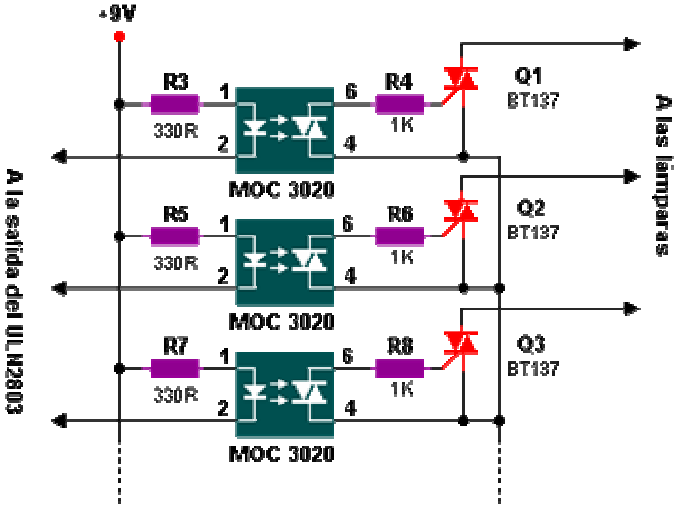


Los interruptores serán conectados a los pines RA0, RA1, RA2, RA3 y un terminal común a +5V, cuando los interruptores están en off la señal que ingresa por estos pines es GND a través de una resistencia de 10k. Yo utilicé un array de resistencias y lo saqué de una placa de las antiguas PC's.

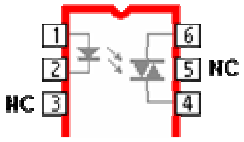
Por otro lado, RB0 a RB7 son las salidas, el tema es que sólo entregan 5V, así que lo levantaremos con un ULN2803 que es un Array de 8 transistores, solo hay que tener en cuenta que este integrado es de colector abierto, o sea que tiene las salidas invertidas, es decir que cuando el micro envíe un 1, el ULN enviará un cero, por lo tanto los LED's para monitorear las salidas deben ir con el ánodo al positivo de la fuente (con su debida resistencia por supuesto...!!!) y el cátodo al ULN, así



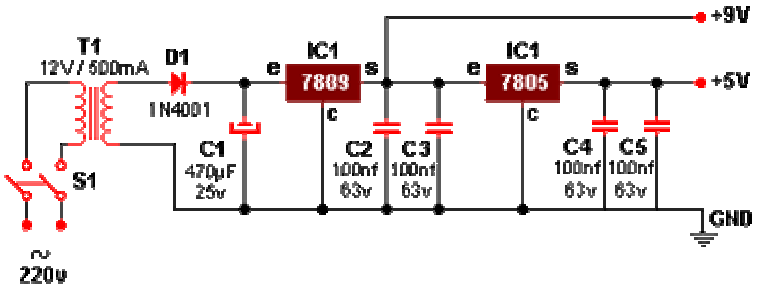
Desde la misma salida del ULN puedes conectar el circuito de potencia que bien puede ser con transistores o con opto triac's como en este caso, no olvides la inversión del ULN please...!!!, esta es la etapa de potencia que yo utilizo en este secuenciador, la resistencia que va al opto puede ser de 220R o 330R



Claro que sólo representé 3 de las etapas de potencia, faltarían las otras 5 pero son lo mismo, el MOC3020 es un opto triac, muy fácil de conseguir, el Triac de potencia es un BT137 en el PCB figura BT136, la disposición de pines es la misma, pero prefiero el BT137 que si mal no recuerdo es de 6 amperios, bien, la disposición de pines del MOC3020 sería la siguiente...

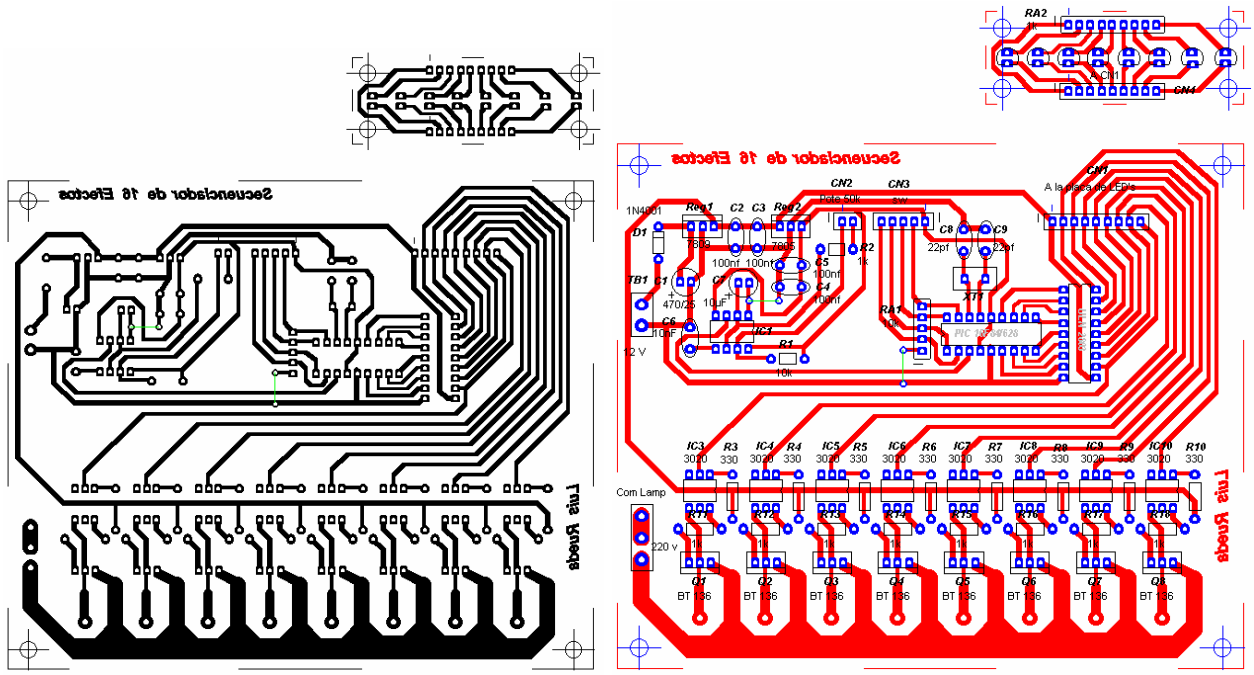


La fuente de alimentación es una muy sencilla con un solo diodo pero con 2 reguladores de tensión, un 7805 para el micro y un 7809 para el ULN2803, los LED's y la Etapa de Potencia, así no tendremos problemas en las salidas.



Esquema de la fuente e imagen del IC 78XX

Y como no podía faltar aquí les dejo un par de imágenes del PCB que por supuesto no es en tamaño real, sólo son imágenes...



Pueden descargar el proyecto completo desde el enlace de la web que se incluye al final del libro.

Secuenciador de 32 Canales Controlado por PC

Hace un tiempo atrás, que hemos estado trabajando en este tema a fin de corregir algunas dificultades que se presentaron en este proyecto que originalmente fue descargado de Discolitez, les recomiendo visitar esta web, por todas las novedades que tiene actualmente.

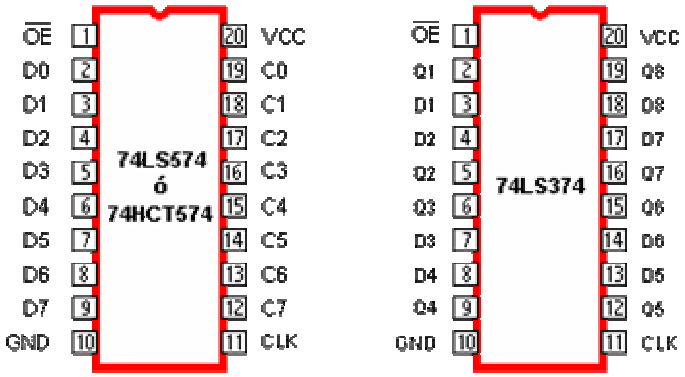
En propuesta de uno de mis alumnos asumimos el desafío, y los resultados los estamos desfrutando desde hace 3 años.

La idea era controlar de forma individual el encendido de 32 lámparas, hasta que al fin lo logramos, aunque le hicimos un par de modificaciones.

Para empezar, modificamos la interfaz. Debido al cruce de pistas en el diseño del PCB, buscamos un nuevo integrado que cumpla con las mismas características que el original, y apareció... En la actualidad contamos con 2 circuitos similares y controlamos 64 canales con dos puertos paralelos, la velocidad del soft es muy alta y funciona tanto en Win'98 como en Win'Me, con WinXP no obtuvimos buenos resultados, al menos por ahora.

Actualmente estamos en la etapa de diseño de nuestro propio soft, y en cuanto salga tendrán noticias de ello.

Nuestro primer circuito fue construido con un integrado 74LS374 y que todavía sigue funcionando, este integrado posee las mismas características que el 74LS574, lo único que los diferencia es la disposición de pines, observa la imagen de la derecha.



La nueva interfaz fue construida con 4 integrados 74HCT574, se trata de un Flip-Flop octal tipo-D, es decir que tiene 8 flip-flop tipo-D *aquellos que cambian con cada pulso de reloj*, también cuenta con salida 3-state, es decir que las salidas pueden adoptar tres estados, (alto, bajo y de alta impedancia).

INPUTS		OUTPUTS	
OC	CLK	D	C o Q
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

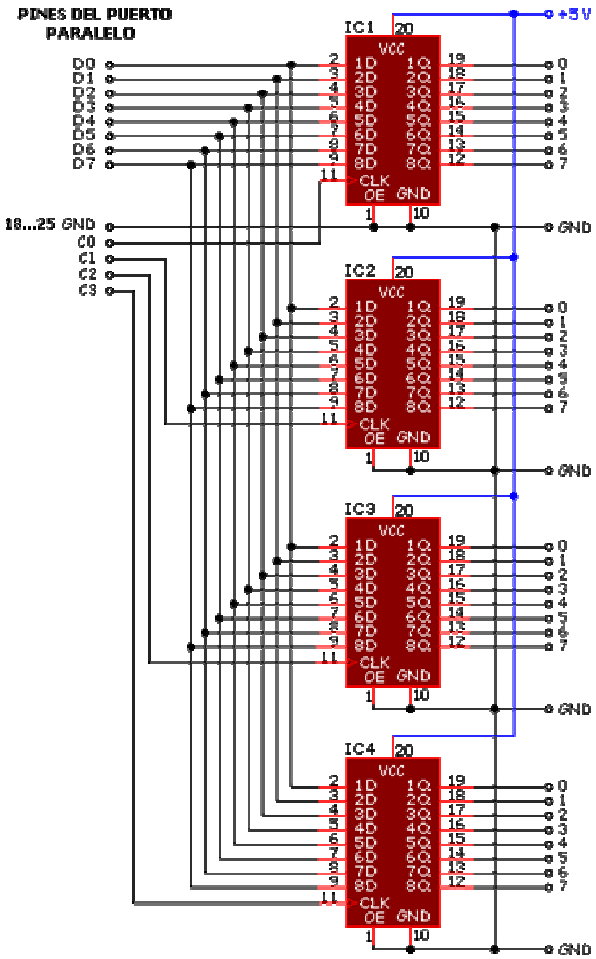
Bien, en ambos integrados, el dato en las entradas se transmite a las salidas (**C** o **Q**) en cada transición de pulsos de reloj de bajo-a-alto presente en la entrada CLK (o entrada de reloj), veamos como lo hace...

Cuando la entrada CLK es alta, las salidas Q toman los niveles de los datos D de las entradas, Cuando CLK pasa a nivel bajo el Flip-Flop retiene el nivel de los datos de salida, y los mantiene en ese estado hasta que vuelve a cambiar de nivel, momento en el cual transmite el nuevo dato que se encuentra en las entradas.

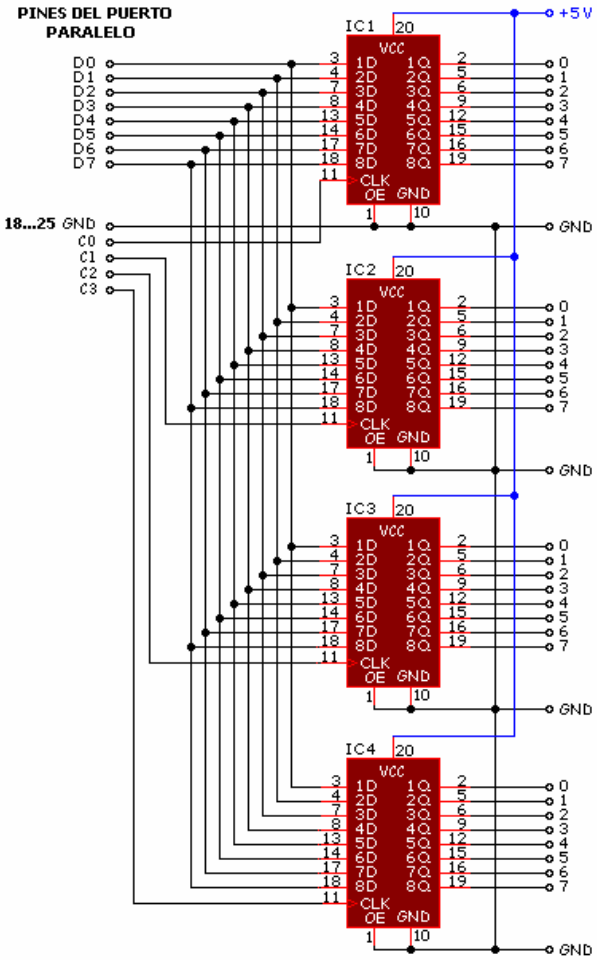
El terminal (OC) puede usarse para poner las 8 salidas en un estado lógico alto o bajo, o de alta-impedancia. OC no afecta el funcionamiento interno de los latchs o flip-flops. Es decir, los datos viejos pueden retenerse o los nuevos pueden entrar en los datos, incluso mientras las salidas están en off, si este pin pasa a nivel alto las entradas de dato y de reloj quedan sin importancia y las salidas Q quedan en un nivel de alta impedancia. Analiza la tabla de verdad de estos integrados...

El esquema del circuito es muy sencillo, y consiste en la implementación de 4 integrados similares, que comparten el bus de datos del puerto paralelo, cada uno de estos integrados se habilitan mediante el pin CLK conectado a los pines de control del puerto paralelo de tal forma que el esquema del circuito es el siguiente...

IC 74HCT574



IC 74LS374

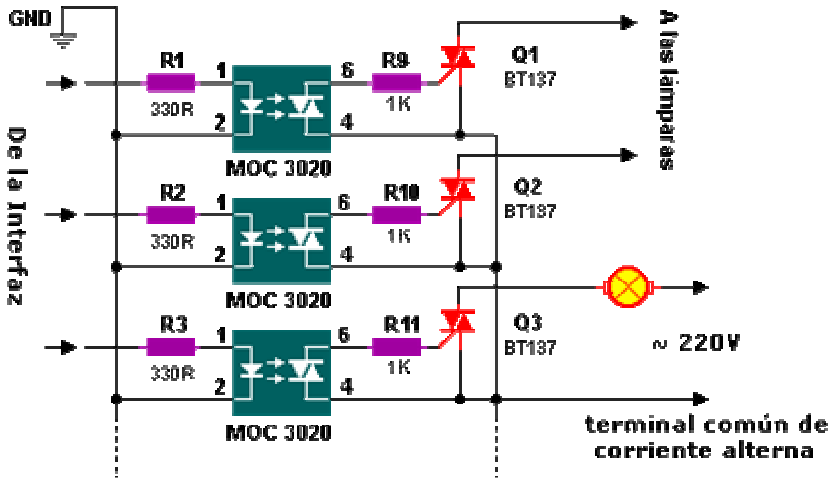


En este esquema se representa el circuito con 4 integrados 74LS574 (los cuales no pude conseguir, en lugar de ellos utilicé los 74HCT574), por otro lado y para que compares, también incluí el mismo circuito pero para el 74LS374, como verás lo único que cambia es la distribución de los pines del integrado (entradas y salidas), la fuente de alimentación utilizada es la misma que se describe en uno de los primeros proyectos, construida con un regulador de tensión 7805, cabe aclarar que este circuito no posee protección para el puerto paralelo del PC, por lo que se debe tener mucho cuidado y comprobar el circuito antes de conectarlo al PC.

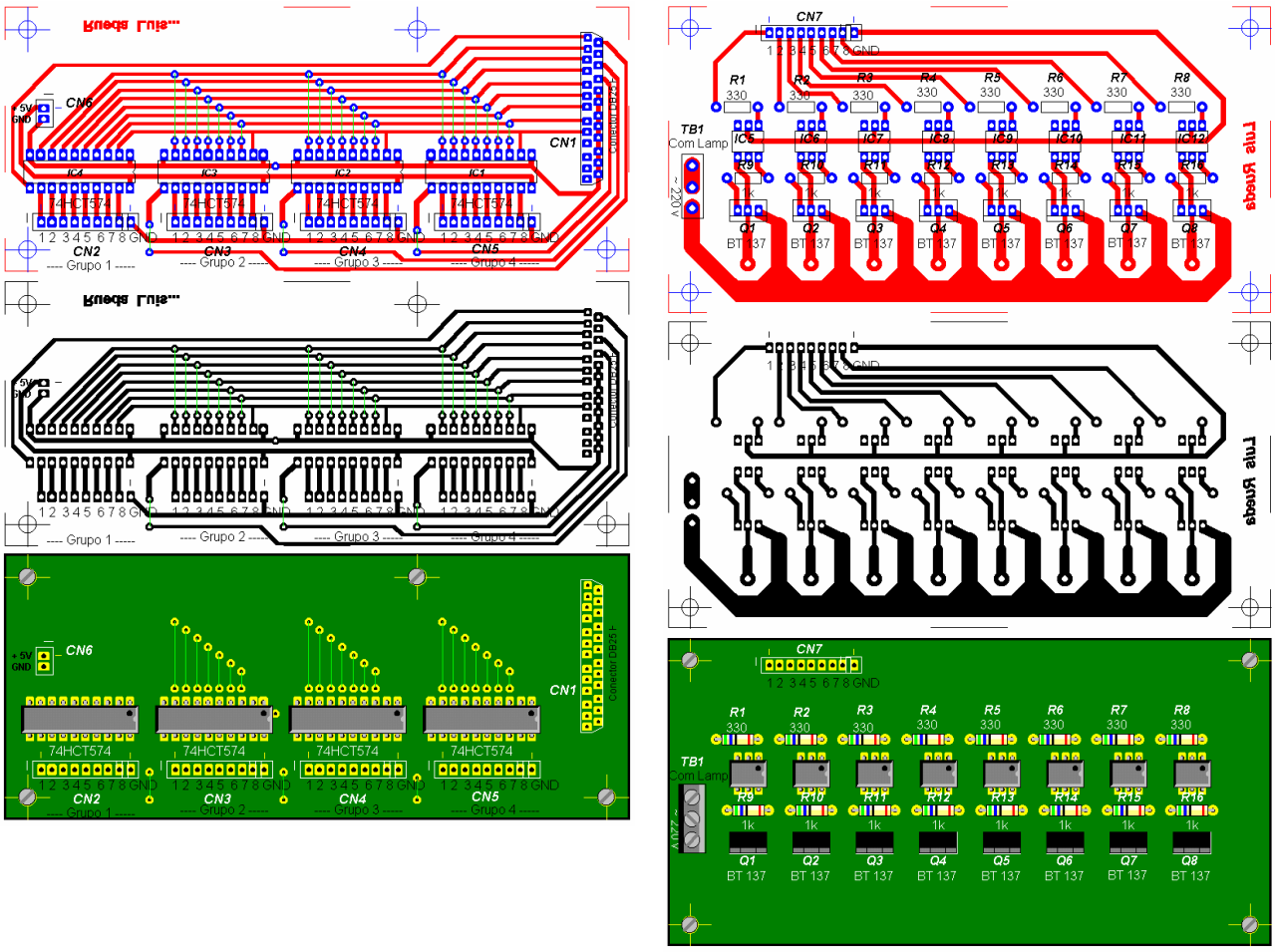
Yo te diría que primero armes el circuito en una placa de pruebas (protoboard) utilizando Diodos LED's para la salida y conectando el cátodo de los mismos a GND de la fuente, y una vez verificado recién montarlo en un impreso, con zócalos claro está.

La etapa de potencia puede ser la misma que utiliza el proyecto **CQPIC**, con la diferencia que esta vez las entradas que van al opto lo harán por el ánodo del mismo y el cátodo irá unido a GND, así...

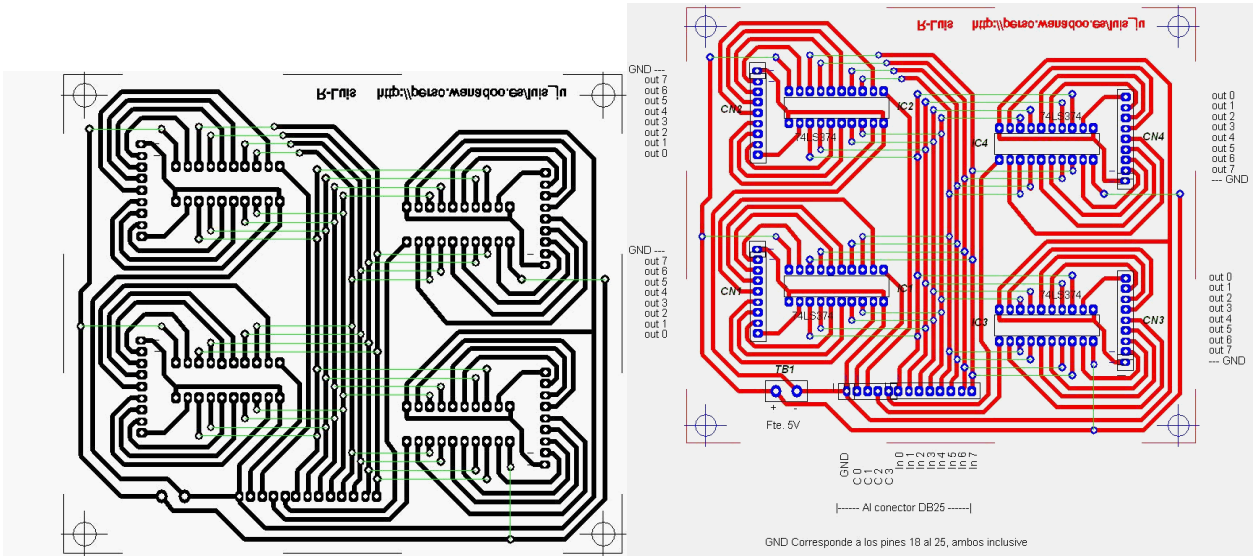
En esta etapa de potencia también incluí una lámpara para que te guíes como conectarlas, todas estas comparten un sólo terminal común unido a la corriente alterna de 220V.

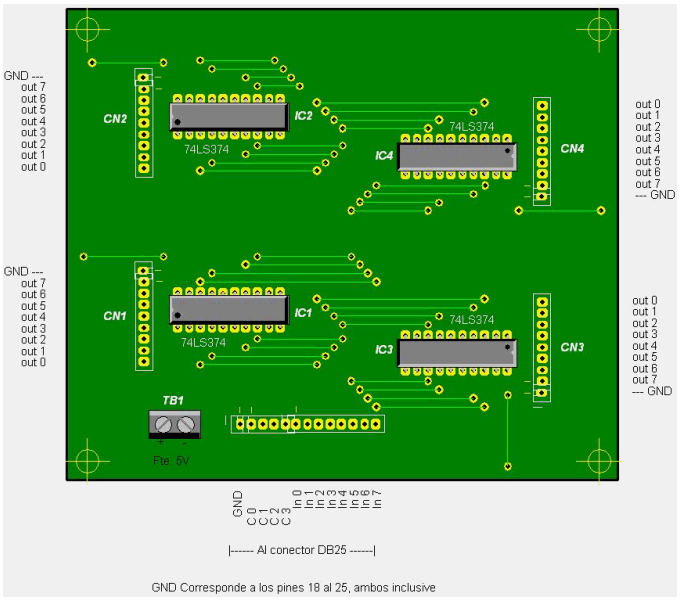


Aquí tienes un par de imágenes del pcb tanto para la interfaz como para la etapa de potencia, esta interfaz está diseñada para el integrado 74HCT574.



La primera versión del circuito que fue montada con 4 integrados 74LS374 la puedes ver desde aquí...

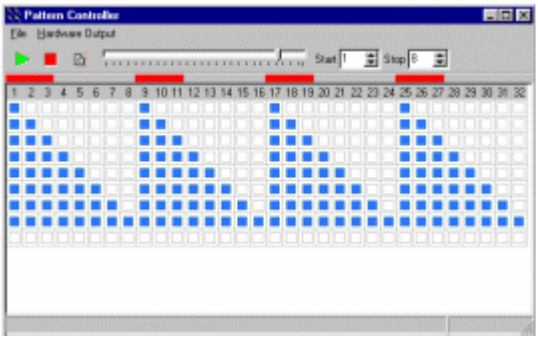




Para armar el cable del puerto utiliza dos conectores macho, y únelos pin a pin, es decir el 1 con el 1 el 2 con el 2 etc. los pines 18 al 25 deben ir todos unidos a un sólo cable.

Lo más importante aquí es que para ver este secuenciador funcionando necesitarás el soft, que puedes descargarlo directamente desde esta url: http:// discolitez.com

Una vez lo tienes, lo descomprimes y lo ejecutas, no requiere instalación, más fácil todavía, luego carga el efecto de luz que más te guste, y le das a play, aquí hay uno efecto ejecutándose, observa...



Por cierto no olvides descargar **DLO_DELUXE32.DLL** se trata de la dll que controla la transmisión de datos al puerto y que también está disponible en el mismo sitio.

Para cargar la dll apunta al menú **Hardware Output --> Load DLOP...** y selecciona la dll, luego configura el puerto, te diriges a **Hardware Output --> Configure DLOP...** y selecciona **$378** en la lista desplegable (que es justamente el puerto paralelo, "**de impresora...!!!**"), finalmente le das a **Ok** y ya está...

Yo creería que está completo solo resta dejar todo en tus manos así que aquí lo tienes, y si algo se me olvida... a consultar, que lo importante es acabar el proyecto y que funcione.

Lo único que no se incluye en este archivo zip es el circuito de fuente que ya está descrito en proyectos anteriores, ni el soft, por una cuestión de derechos de autor, pero tienes toda la posibilidad de bajarlo desde su propio sitio, más no se puede pedir.

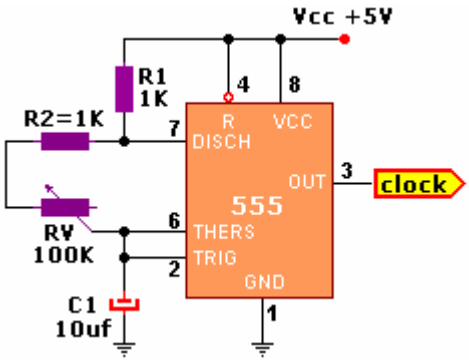
Bien, espero lo disfruten una vez acabado el proyecto.

Secuenciador de 8 Canales y 4 Efectos

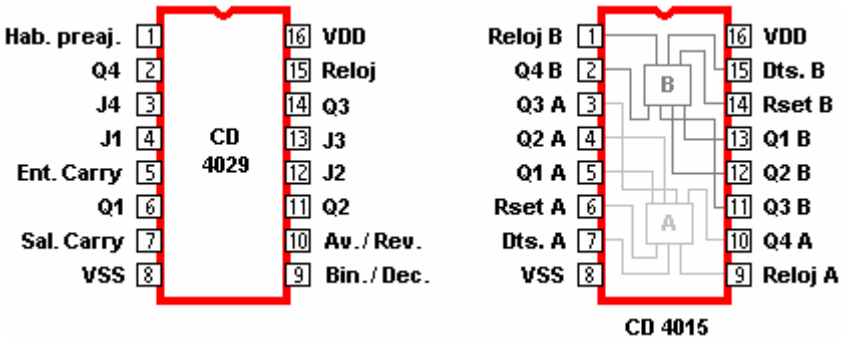
A pesar de que muchos a estas alturas preferimos un secuenciador con PIC's, hay quienes prefieren hacerlo con un integrado común, para no liar con las instrucciones de assembler o por no tener un grabador de PIC. En mis inicios, desesperado por un secuenciador distinto al del 4017, me topé con éste, lo probé y luego de unos arreglos lo armé y aquí se los dejo, ya que muchos lo estuvieron pidiendo.

Lo más común en todos los secuenciadores es la etapa de potencia por lo que no la describiré.

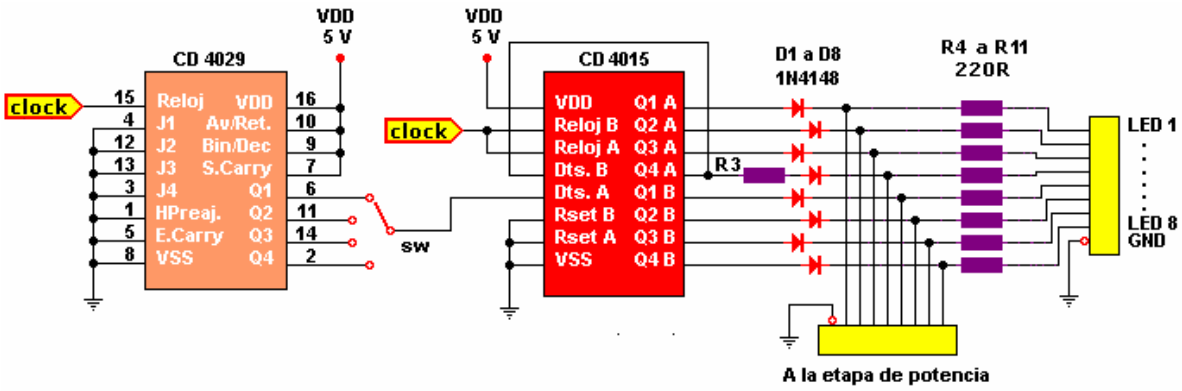
Bien, comencemos, se trata de un sencillo secuenciador de 8 canales y 4 efectos, los efectos son seleccionados por una sola llave selectora de 4 posiciones, y como todo secuenciador requiere un timer que genere los pulsos..., que más...!!!, vamos por un 555 así...



Los pulsos que envía el 555 son recibidos por dos integrados, uno de ellos es un contador binario (CD4029) y el otro es un doble registro de desplazamiento de 4 bits (CD4015) ambos muy económicos y fáciles de conseguir, respecto a los datos del integrado CD4029 y sus modos de conexión, pueden verlo en la sección de IC-Datos CD4029, de todos modos les dejo una imagen de estos integrados.

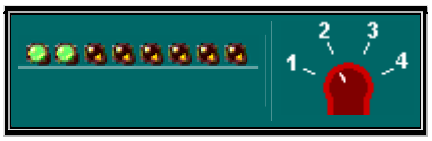


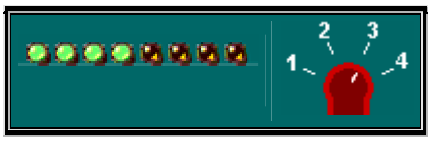
El circuito en cuestión es el que sigue...

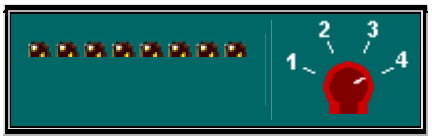


El funcionamiento es relativamente simple, el integrado CD4029 está configurado para trabajar en modo contador binario, los 4 bits de salida obviamente tienen distinta duración según la cuenta avanza, recuerda que es 0000, 0001, 0010, 0011, y así sucesivamente.

Bien, uno de esos bits (según la posición de sw (la llave selectora)) serán recibidos por la entrada DATOS_A del primer registro de desplazamiento del CD4015, ahora bien, la entrada de reloj tomará ese dato en cada pulso y lo enviará a la salida correspondiente (Q1, Q2, Q3 y Q4), observa también que la salida Q4A está unida a la entrada DATOS_B, éste es el segundo registro de desplazamiento del CD4015, este puente hará que se repita el desplazamiento del primero, como que en lugar de tener dos registros de desplazamiento de 4 bits separados tengamos uno sólo de 8 bits, lo más extraño es la resistencia R3 ya que es la única salida que lleva esa resistencia, pues sucede que si no estuviera, el LED consumiría toda la corriente que debería ingresar a DATOS_B, bueno, la resistencia soluciona este inconveniente.

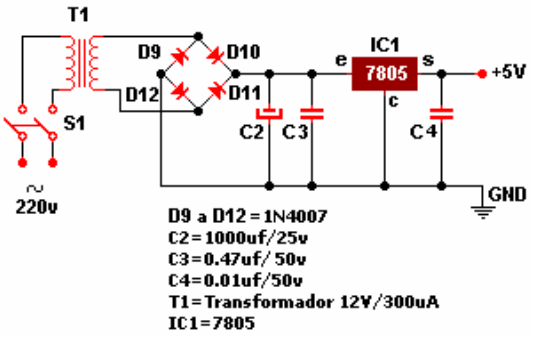




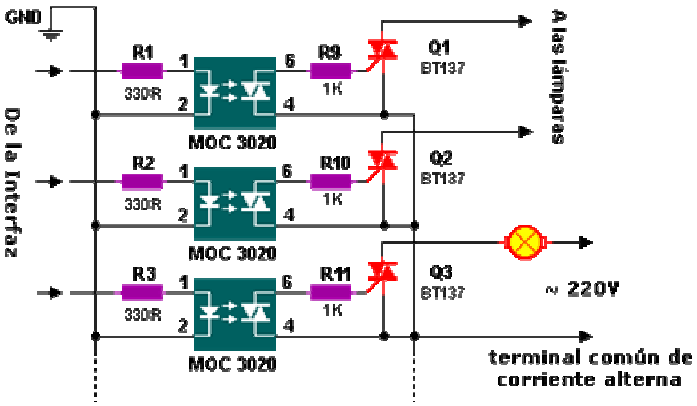


Ahora sí, hablemos de los efectos, si la entrada DATO dura un pulso de reloj, tendrás en la salida 10101010 y eso es lo que se desplazará, si la entrada DATO dura 2 pulsos tendrás 11001100, si dura 4 tendrás 11110000, y por supuesto, si dura 8 tendrás 11111111, pero mejor lo veamos con un par de imágenes

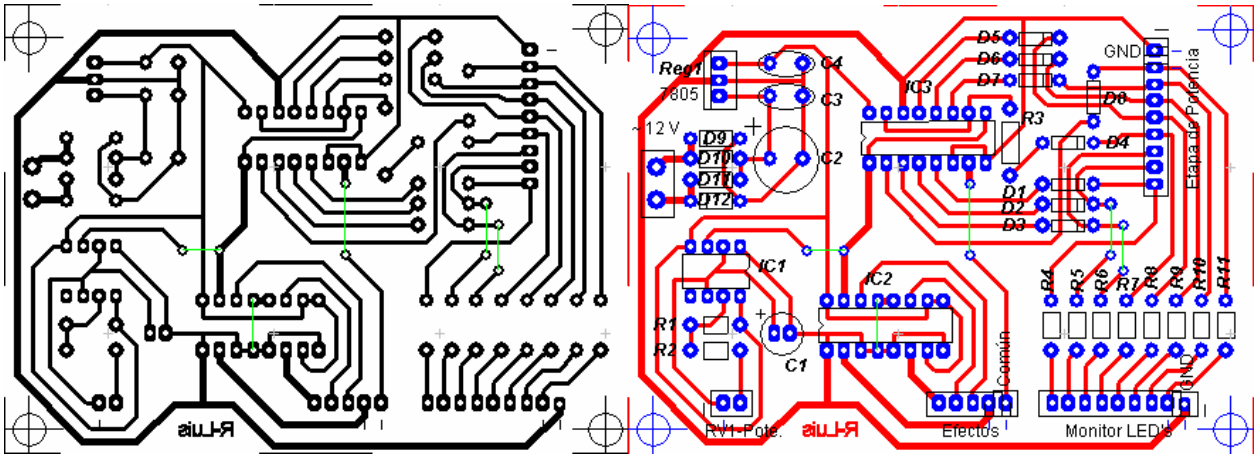
El último efecto es el que tanto se estuvo pidiendo en el foro y que también lo hicieron por correo, ahora ya lo tienen. La fuente de alimentación es de 5V. lo típico, como la que tenemos en todos los proyectos, algo así...

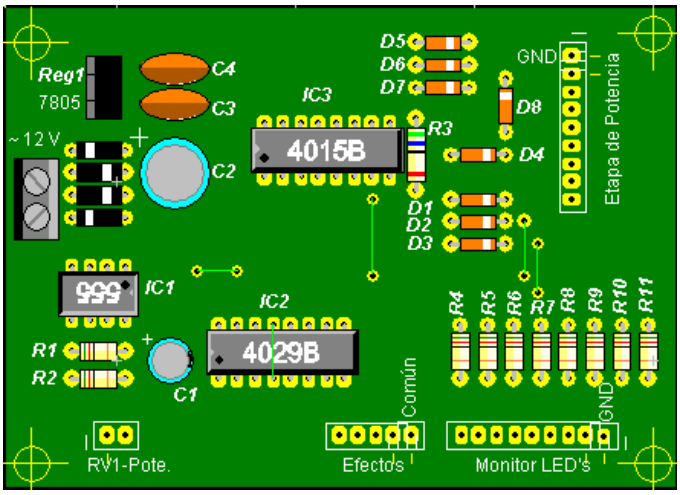


La etapa de potencia también puede ser una de las que vimos anteriormente o utilizar transistores si lo deseas, aquí te dejo un esquema. Sobre todo mucha precaución al trabajar con tensiones elevadas.



Aquí tienes unas imágenes del pcb, claro que no figura la etapa de potencia, pero sí un conector para que le agregues en una placa distinta, al igual que la placa para el monitor de LED's.





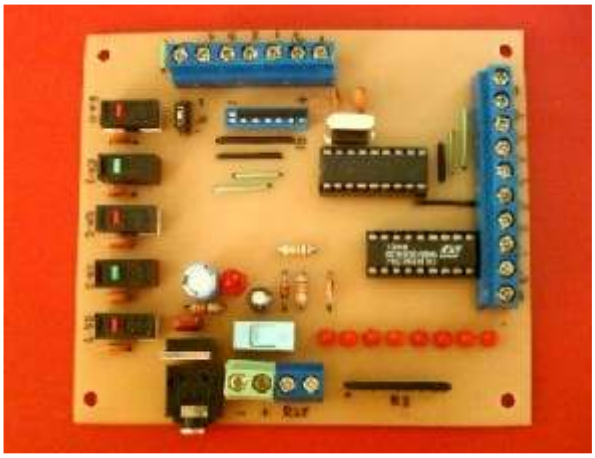
La verdad..., es una maravilla de proyecto.

Entrenador para el PIC 16F84

Existe una gran variedad de entrenadores en toda la red, a los cuales se suma el mío, esto no significa que deban montarlo, pero si de algo les sirve, pues, aquí se los dejo.

El circuito es muy sencillo, y está construído con la intención de ser utilizado en conjunto con otra u otras placas, montadas sobre ellas mimas.

La idea surge con la finalidad de introducirnos en la construcción de minirobots, más bien conocidos como microbots, puesto que desde el inicio de este libro, quería darle esta orientación, aunque luego me fuí por cualquier lado para que aprendan de todo en electrónica. Bien, la que sigue es una imagen del entrenador. Eso sí..., no me vayan a criticar los puentes, se puede hacer mejor pero este es así.



Ahora veamos que es lo que tiene este entrenador y cómo utilizarlo.

Fuente de alimentación:

Como la idea es utilizarlo con baterías, no le puse diodos de rectificación, sino más bien un regulador de tensión (**7805**) para mantenerlo a 5 volts. Ahora bien, posee dos conectores, uno de ellos es para cuando lo utilizemos como entrenador, ya que por lo general disponemos de una fuente rectificada y una ficha tipo plug, el otro conector es simplemente una bornera (aquella de color verde, justamente para identificarla), allí se encuentran marcados los polos de conexión, para así evitar errores, todo esto lo puedes ver en la siguiente imagen...

Me olvidaba..., el LED que se encuentra en la fuente, es para saber si realmente está llegando tensión a la placa, y así no tenemos necesidad de estar testeando para hacer esta prueba.

Circuito de Reset:

Del mismo lado y junto a la fuente se encuentra el circuito de **Reset** del micro, que por lo general lo utilizamos en nuestro entrenamiento, es por eso que cuenta con un micropulsador, (que lo saqué de un viejo mouse, quien dijo que hay que tirarlos...???)

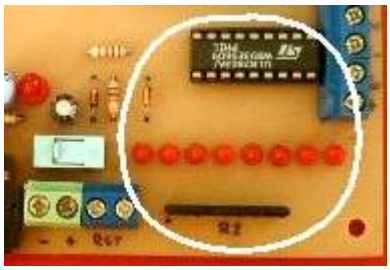
Cuando comencemos nuestros proyectos con microbótica, seguramente resetearemos el micro desde algún sensor, o algo por el estilo, asi es que le agregué una bornera para cuando llegue ese momento.

Te preguntarás como utilizar la bornera, pues muy sensillo, sólo debes unir los dos bornes, y habrás ocasionado un reset en el micro

El Puerto B:

De buenas a primeras, tienes la posibilidad de probar tus programas haciendo uso de las entradas y salidas que desees, por ejemplo, se ha previsto al puerto B del micro, con un integrado ULN2803 y 8 LED's, acompañados de un Array de resistencias **R2**, puesto que por lo general, este puerto se utiliza para salida.

Por ejemplo, si envías a este puerto el valor 00000011 se encenderán los dos primeros LED's. Cabe aclarar que el ULN muere allí, en los LED's



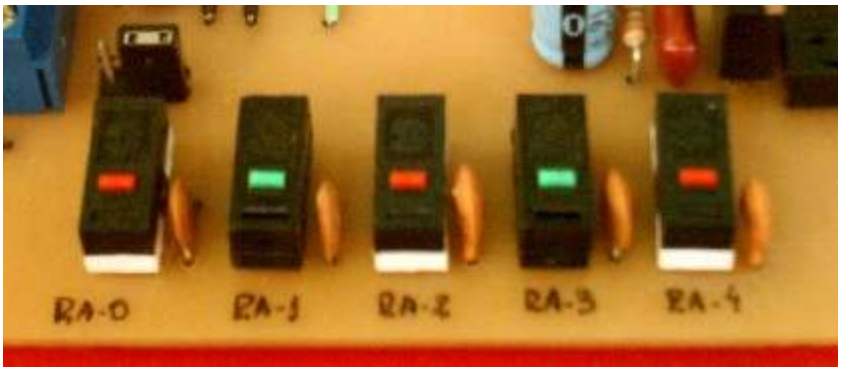
De todos modos, del lado del ULN se encuentra una colección de borneras las cuales corresponden al puerto B del micro y reflejan el estado de los pines del PIC16F84 (**NO del ULN**), por si deseas utilizar estas señales para cualquier otra tarea, como por ejemplo, la utilización de Displays, u otros dispositivos.

Por si fuera poco, los extremos de la bornera poseen los polos de la fuente de alimentación de la placa (**0v** y **5v**), por si deseas introducir señales por estos pines, así no estás metiéndole cualquier cosa al micro, sino los niveles de tensión que corresponden.

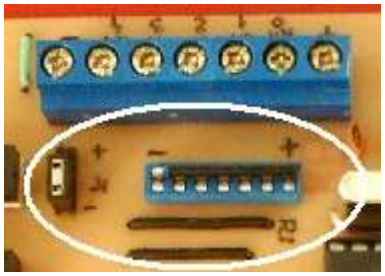
El Puerto A:

Al puerto A, le puse unos pulsadores (sacados del mouse), están colocados de tal modo que tengas la posibilidad de seleccionar la polaridad de los mismos, mediante un jumper de tres puntos (**J1**), donde el punto medio es el común de los pulsadores, y sus extremos son los polos positivo y negativo, esto nos dá la libertad de elegir el tipo de señal que se enviará al micro por los pines de este puerto.

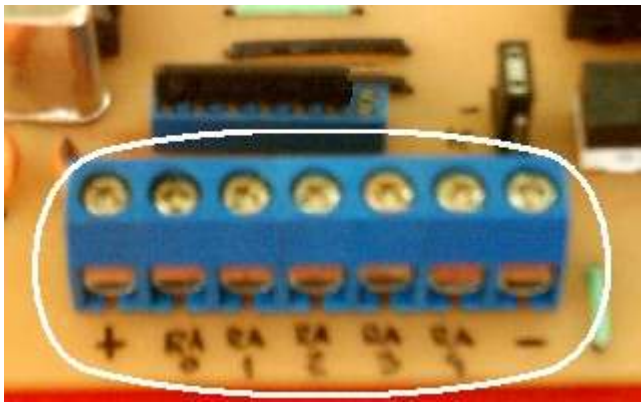
Los capacitores junto a los pulsadores se implementaron para evitar la inestabilidad de los mismos, aquello que conocemos por rebotes eléctricos.



Notarás también, que del lado de las borneras del puerto A, se encuentra un array de resistencias de 10k, y que le sobra un pin de conexión, pues bien, uno de los extremos es positivo y el otro negativo, te preguntarás, por que...??, sencillo..., si al activar los pulsadores envian cero al micro, cuando reposen deberán estar en uno. Esto significa que si el jumper (**J1**) de los pulsadores está unido al polo negativo, el array deberá estar unido al positivo, y al revés, cuando el jumper de los pulsadores esté en positivo el array deberá estar en negativo, y así al activar el pulsador enviarás un uno lógico.



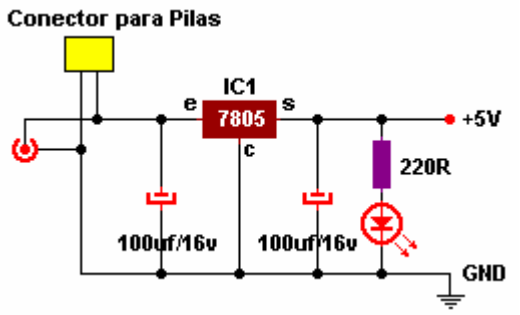
También puedes notar en esta imagen, que se encuentra una bornera correspondiente al puerto RBA, esto es para cuando no utilices los pulsadores, en ese caso, la señal que ingrese por el puerto A vendrá seguramente de sensores de un microbot o de cualquier cacharro que estemos haciendo, por la misma razón dejé los extremos de la bornera con los polos de la fuente de alimentación, y así no nos preocupamos por la alimentación de esos sensores.



Bien mis queridos amigos, sólo resta dejarles los esquemas del circuito y el PCB para quienes deseen montarlo.

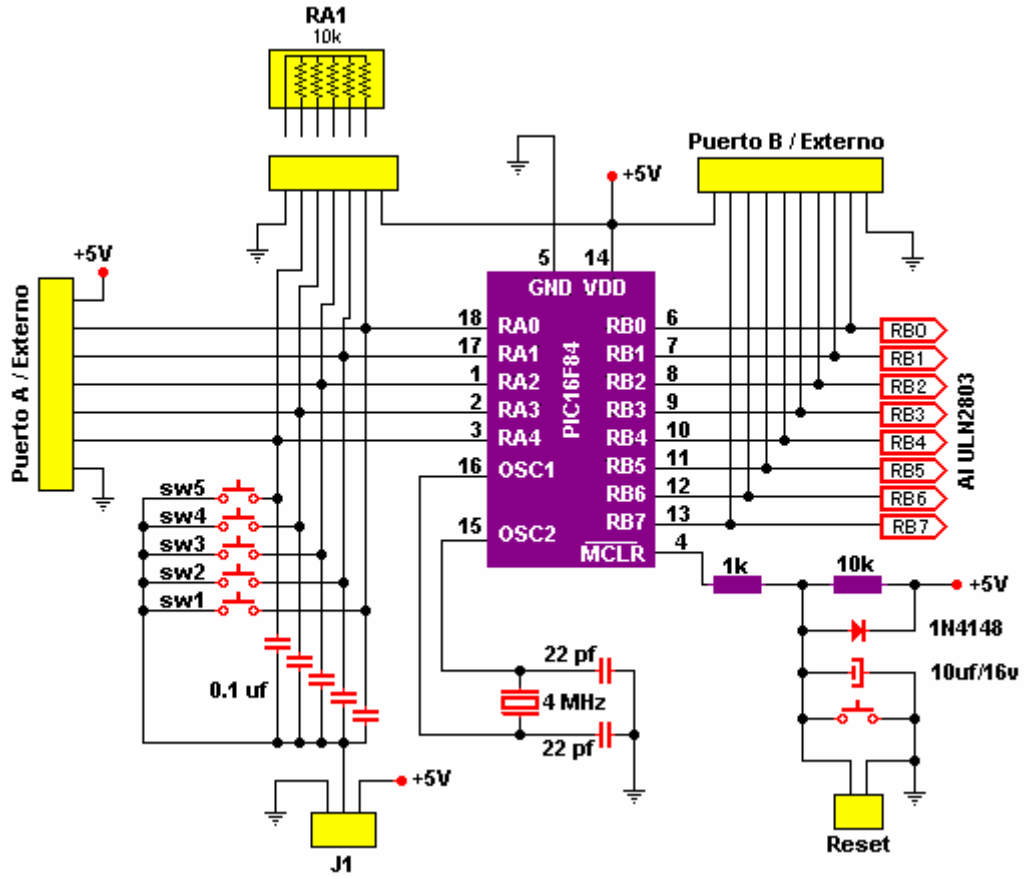
Fuente de Alimentación:

Como verás la fuente de alimentación consiste en un regulador de tensión 7805 y un par de capacitores, una ficha tipo jack, y un conector por si debieras hacerlo con cables, el LED es simplemente para verificar el estado de la fuente.



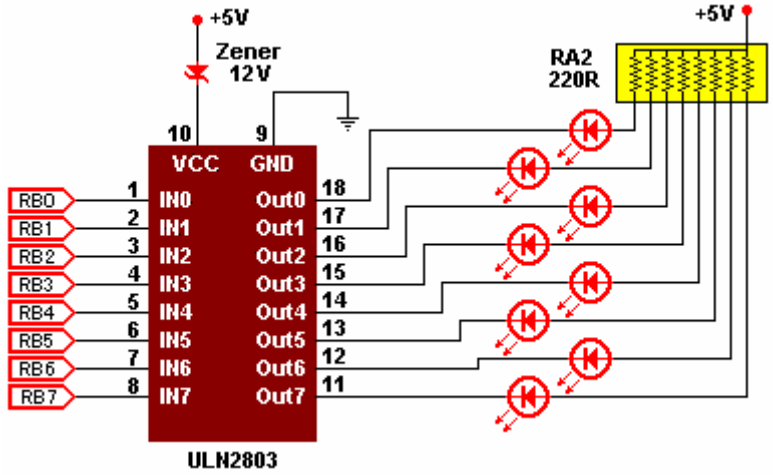
El Microcontrolador:

Aquí hay varias cosas para menciones, pero vamos por partes.**El circuito de Reset:** por si fuera necesario reiniciar el micro, cuenta con un pulsador o bien es posible utilizar la bornera para activarlo externamente (podría ser con sensores).**Las Salidas RB0-RB7:** son para monitorearlas por medio del ULN y 8 LED's.**Puerto B / Externo:** es una bornera, que sirve de comunicación con el exterior.**RA1 y J1: RA1** Es un Array de Resistencias, que tiene un terminal común a todas, J1 es un Jumper.**Bien, se pueden presentar dos casos:1)-** Que la señal de entrada sea **Positiva** (J1 conectado a +5V) en cuyo caso RA1 deberá tener el común conectado a GND.**2)-** El caso opuesto, que la señal de entrada sea **Negativa** (J1 conectado a GND), en ese caso RA1 deberá ir a (+5V).



Monitoreo de Salidas

Consiste en un simple ULN, podría haber sido otro pero bueno, es lo que hay



Advertencia:

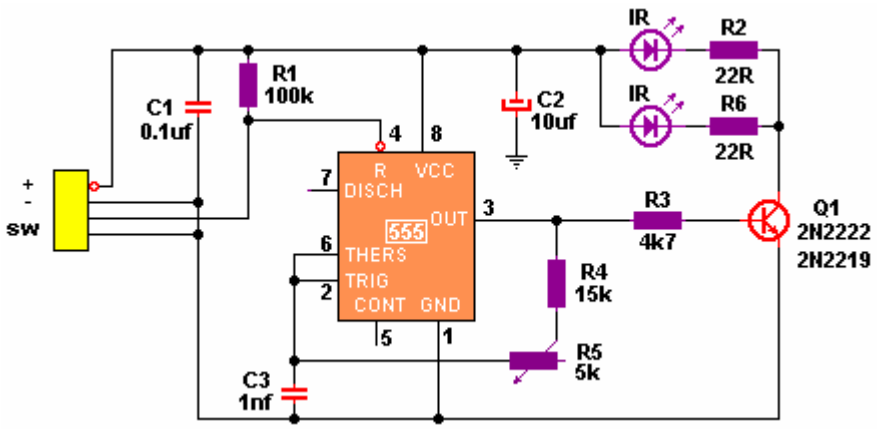
Quizás le encuentre una fallo, o puedan mejorarlo, o agregarle algo nuevo, nose, no se me ocurre en estos momentos, en fin, aquí queda, en revisión. El enlace para descargar los archivos de este proyecto lo encontrará al final del libro.

Emisor y Receptor Infrarrojo

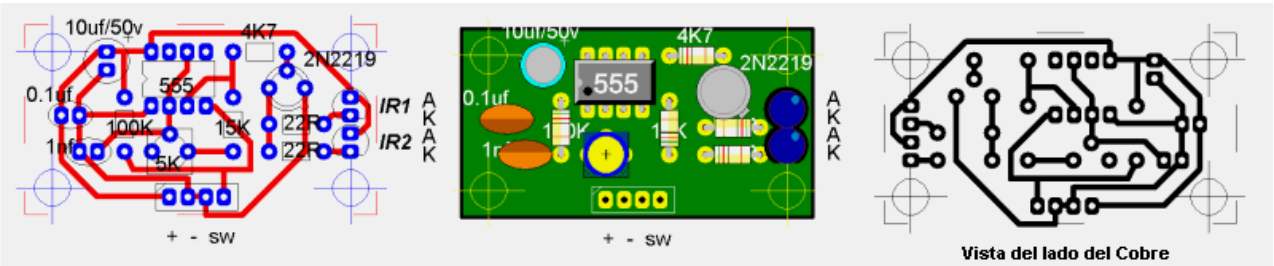
Espero que hayan leído la sección de Sensores donde ya se comentó demasiado al respecto, por lo que no habrá una introducción teórica, sino una aplicación de esa teoría en la construcción de módulos que quizás nos sean de utilidad en algún momento, aunque más no sea por puro hobby.

Bien, para dar inicio a este apartado, te mostraré el primer módulo...

El esquema del circuito es muy sencillo, aquí se incorporaron dos emisores IR, para futuras aplicaciones, es más los espadines los puse con la idea de montarlo en cualquier otro circuito, sin necesidad de atornillar, ni nada por el estilo...



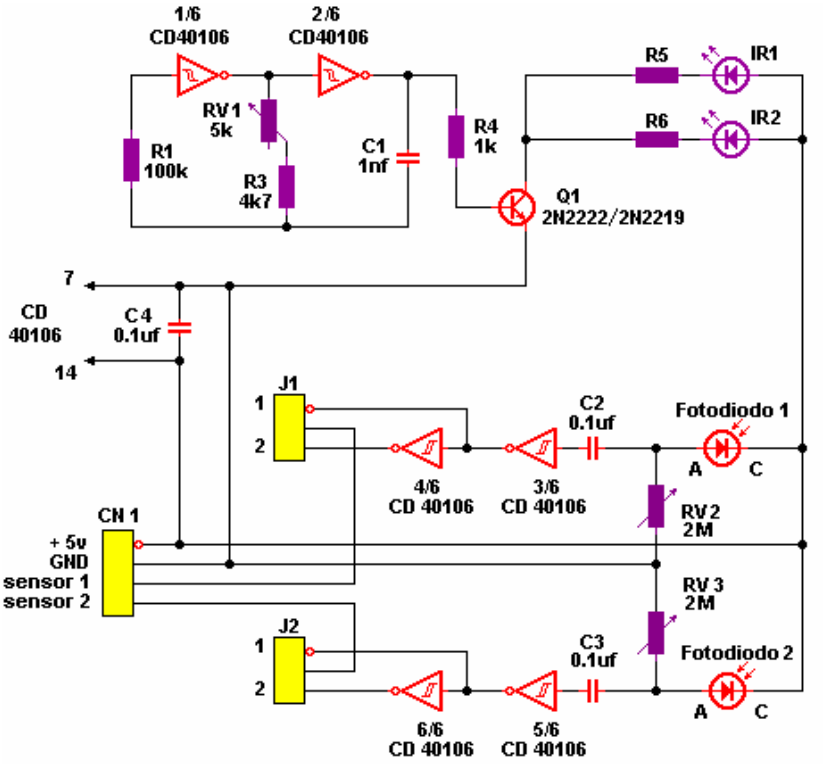
Respecto al alcance que se puede lograr con este circuito, depende de la tensión de trabajo, y uno que otro ajuste en el receptor, por ejemplo con una batería de 9V alcancé medio metro +o-. El PCB para este circuito lo puedes ver aquí...



El que viene a continuación surgió con la idea de hacer un seguidor de líneas, éste está basado en el conocidísimo integrado CD40106, el módulo incorpora emisor y receptor, más una salida

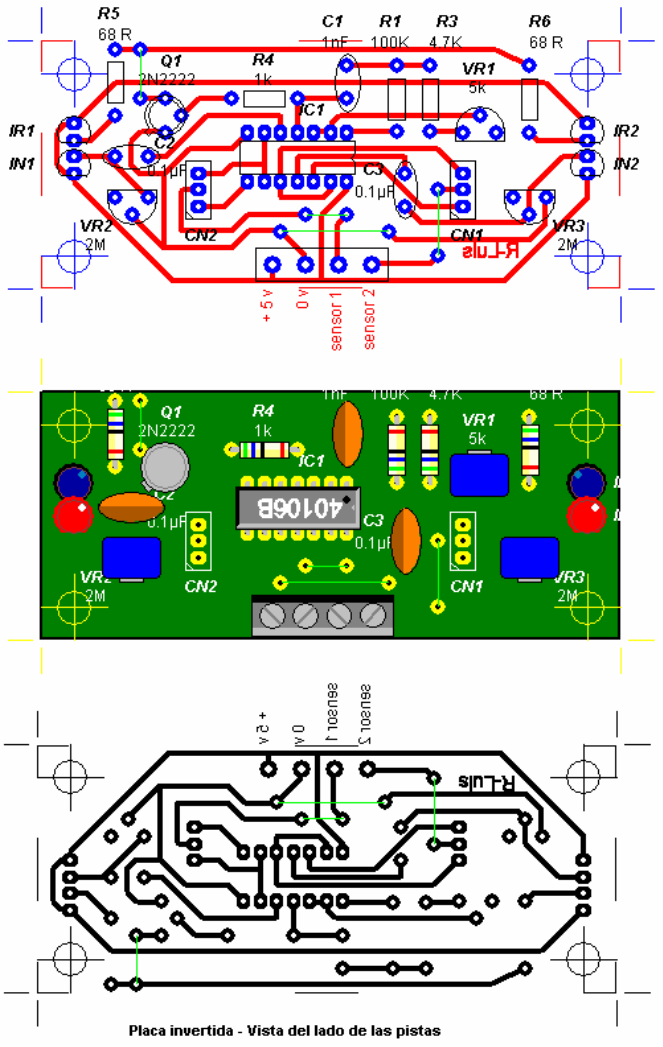
para cada sensor el derecho y el izquierdo que refleja su estado en forma lógica es decir con un 0 o un 1, interesante no...???

Bien, el esquema del circuito es el que tenemos a la dercha.



Bueno, no se asusten, la placa de circuito impreso final mide unos…. 9cm x 4cm, algo que quiero destacar, es que tanto los LEd's IR y los fotodiodos, fueron acomodados en la placa de tal forma que en lugar de ellos pueda ir un conector de 4 pines, y de esta manera, el par Emisor-Receptor será externo al módulo.

Lo que sigue, es el PCB...



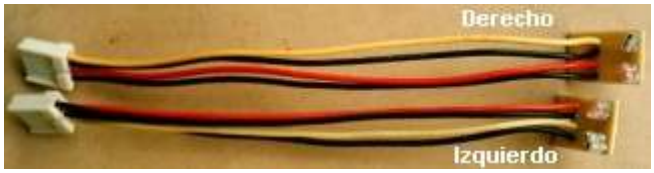
Placa invertida - Vista del lado de las pistas

Y ahora, vamos a describir algunos detalles de esta placa, y como podemos aprovecharla, mas que nada, las opciones de configuración que ésta tiene, pero antes necesitamos una imagen real de este módulo...

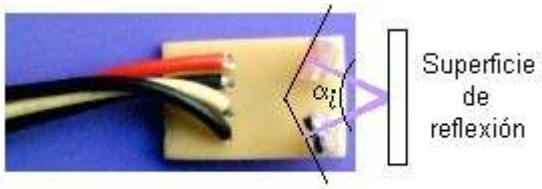
Vamos por partes...

- RV1 (que aparece allí como REG. IR) Es la resistencia variable (Preset) que me permite regular la frecuencia del oscilador.
- RV2 (REG1) y RV3 (REG2), son los Preset que regulan la sensibilidad de los fotodiodos conectados a la Izquierda y a la Derecha del módulo respectivamente.
- J1 y J2 son los Jumper de selección del estado lógico que reflejará la salida de los sensores, y como cumplen la misma función en ambos lados, sólo describiré uno...
  Con J en la posición 1, el receptor estará trabajando con un sólo inversor, en esta situación enviará un 1 lógico a la salida siempre que se esté recibiendo la señal del IR
  Con J en la posición 2, la situación será totalmente opuesta, el receptor enviará un 0 lógico a la salida siempre que se esté recibiendo señal del IR.
- La bornera verde es para la fuente de alimentación, que si la utilizarás con un microcontrolador por ejemplo deberá trabajar a 5V.
- La bornera azul es la que envía la información del estado de los sensores que mencionamos anteriormente, del lado de la fuente el estado del sensor izquierdo y el que sigue corresponde al derecho. Creo que esto, será muy útil cuando trabajemos con nuestro bien amado PIC 16f84.

Bueno, no incluí en el PCB la pareja Receptor-Emisor, creo que no es necesario, ya podrás armarlo tú solito, lo que si quiero comentar es que deberás regular el ángulo de incidencia entre el emisor y el receptor, ya que eso dependerá de los componentes (IR y Fotodiodo o fotoransistor) que utilices en este montaje, esto es algo así como armar tu propio CNY70, para que tengas una idea, mira como me quedaron a mí, y no se ven tan mal...



Por cierto y antes que lo olvide, deberás regular el ángulo de incidencia entre emisor y receptor, si prestas atención a la siguiente imagen te darás cuenta a lo que me refiero.



α_i es el ángulo de incidencia

En realidad el ángulo de incidencia es el que forma el rayo de luz con la normal a la superficie de reflexión, pero aquí lo vamos a resumir... Ahora veamos como queda nuestro módulo funcionando con todos sus agregados...

Aquí finaliza esta maravillosa obra.

Palabras finales...

Para comodidad del lector, he agrupado en un solo archivo todos los proyectos aquí descritos, no obstante puede visitar la web de Luis y allí encontrarás todo lo que hay en este libro, mis más sinceras felicitaciones a Luis, tu trabajo es sorprendente.

http://r-luis.xbot.es

Espero que sea del agrado de muchos y que puedan trabajar con todos estos elementos que son fáciles de conseguir.

Reciban mis más Cordiales Saludos...

Enlace para descargar los proyectos de este libro:

Aquí un archivo con todos los proyectos: http://mikroe.es

Aquí encontrará todos los proyectos desde la web de Luis: http://r-luis.xbot.es

Bibliografía:

http://r-luis.xbot.es

Rueda Luis...
luis.ju22@gmail.com
crdato1@yahoo.com.ar

De Luis …..

"Cierta vez leí por ahí... "No guardes en tu cerebro aquellos conocimientos que lograste alcanzar o nadie sabrá de tu paso por estos infiernos", bueno, pues aquí están los míos, y si deciden publicarlo, no olviden mencionar de donde lo sacaron, creo que me lo gané...!!!"

Efectivamente Luis, te lo has ganado y aquí está tu mención, todo esto es obra tuya, yo tan sólo he recogido tu siembra y la transmito para que la gente tenga los conocimientos que tú has enseñado y agradecemos tu labor. Muchas gracias Luis!!!!!

Un saludo desde Barcelona - España

Fin del libro, espero les haya gustado y sobre todo aprendido ya que este era mi propósito. Como nadie es perfecto y yo me encuentro entre la clase humana que no es perfecta, si detectan algún error ruego me lo comuniquen por email (kjm535@yahoo.es) y lo corregiré, al mismo tiempo si hay algo que deba suprimir por tener derechos de autor, me envían la información y lo suprimo, recuerden que las modificaciones serán en otra revisión que la colgaré inmediatamente en la web para su descarga.

Esta obra es gratis, nadie le ha de cobrar dinero alguno por obtenerla.

El que ha escrito este libro basándose en la obra de Luis:

Joan Mengual – Barcelona a febreo de 2008 y finalizado en febrero-2013
http://mikroe.es
joan@mikroe.es
kjm535@yahoo.es